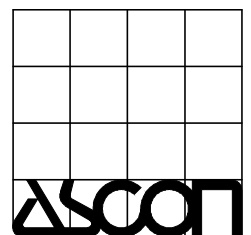
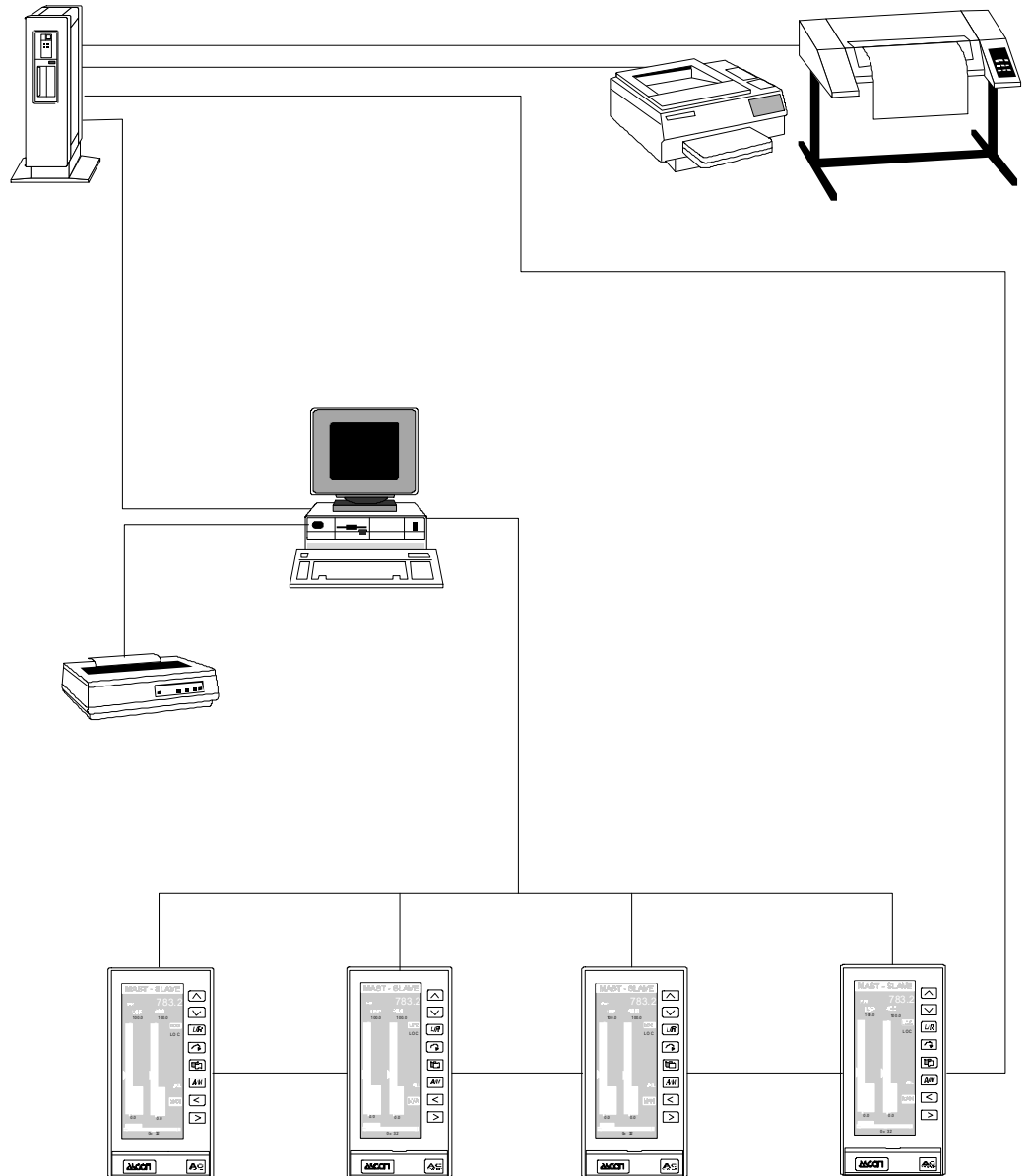
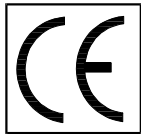


USER GUIDE  
M.I.U. ACCS - 11 / 00 - 07  
Cod. J30 - 304 - 1ACSAC ING



UNI EN ISO 9001



Copyright © 1996 ASCON spa

*All rights reserved*

*No part of this document may be stored in a retrieval system, or transmitted in any form, electronic or mechanical, without prior written permission of ASCON SpA*

*ASCON has used the best care and efforts in preparing this book and believes the information in this publication are accurate. The ASCON Products are subjected to continuous improvement, in order to pursue the technological leadership; these improvements could require changes to the information of this book. ASCON reserves the right to change such information without notice.*

*ASCON makes no warranty of any kind, expressed or implied, with regard to the documentation contained in this book. ASCON shall not be liable in any event - technical and publishing error or omissions - for any incidental and consequential damages, in connection with, or arising out of the use of this book.*

*AC STATION, AC PROGRAPH e AC EDIT, are trademarks of ASCON spa.*

*All the other tradenames or product name are trademark or registered trademarks.*

**ASCON spa**

20021 Ospiate di Bollate ( Milano ) Italy

Via Falzarego 9 / 11 - Tel. ++39-2-333371 - Fax ++39-2-3504243

<http://www.ascon.it>

support@ascon.it

**T A B L E O F C O N T E N T S**

<b>1. INTRODUCTION .....</b>	<b>4</b>
<b>2. MODBUS &amp; JBUS.....</b>	<b>5</b>
2.1 DIFFERENCES BETWEEN MODBUS AND JBUS.....	5
2.2 REFERENCES .....	5
2.3 HARDWARE .....	5
2.4 THE MODBUS PROTOCOL .....	6
2.5 MESSAGE STRUCTURE.....	6
2.6 ADDRESS FIELD .....	7
2.7 FUNCTION CODE .....	7
2.8 CRC16.....	7
2.8.1 <i>How to compute CRC16 in Visual Basic</i> .....	8
2.9 MESSAGES SYNCHRONIZATION.....	8
<b>3. MODBUS FUNCTIONS.....</b>	<b>9</b>
3.1 READ OUTPUT STATUS ( 01 ).....	9
3.2 READ INPUT STATUS ( 02 ).....	9
3.3 READ HOLDING REGISTERS ( 03 ) .....	9
3.4 READ INPUT REGISTERS ( 04 ) .....	10
3.5 FORCE SINGLE COIL ( 05 ).....	10
3.6 PRESET SINGLE REGISTER ( 06 ) .....	10
3.7 FORCE MULTIPLE COILS ( 15 ) .....	11
3.8 PRESET MULTIPLE REGISTERS (16).....	12
<b>4. ERROR HANDLING .....</b>	<b>13</b>
4.1 EXCEPTION CODE .....	13
<b>5. SUPERVISORY SYSTEM .....</b>	<b>15</b>
5.1 DATA ORGANIZATION.....	15
5.2 FLOATING POINT STRUCTURE .....	15
5.3 THE SWAP OF THE FLOATING POINT WORDS .....	16
5.4 LOGIC VARIABLES ( COILS ).....	17
5.5 NUMERIC VARIABLES ( REGISTER ).....	19
5.6 SETTING VALUES AND PARAMETERS OF THE SETPOINT PROGRAMMER.....	25
5.7 SYSTEM VARIABLES .....	25
5.8 READING AND WRITING THE CONFIGURATION .....	26
5.9 AN EXAMPLE OF CONFIGURING AND PROGRAMMING THROUGH THE RS 485 .....	27
5.10 MENU READING.....	32
5.11 REAL READING .....	33
5.12 STRINGS READING .....	33
<b>6. STATIC AND DYNAMIC MODBUS / JBUS ADDRESSES.....</b>	<b>35</b>
6.1 STATIC ADDRESS.....	35
6.2 DYNAMIC ADDRESS .....	35
<b>7. ARCNET .....</b>	<b>37</b>
7.1 HARDWARE .....	37
7.2 ARCNET PROTOCOL FOR SERIES AC CONTROLLER.....	38
7.3 PROTOCOL TYPE A .....	38
7.4 TYPE B PROTOCOL.....	39
7.5 ERROR HANDLING .....	40
<b>8. AC CONTROLLER CALIBRATION THROUGH THE SERIAL PORT .....</b>	<b>41</b>
8.1 OUTPUT CALIBRATION .....	42
8.2 JBUS ADDRESSES IN TEST MODE .....	43
8.3 FUNCTIONS AVAILABLE .....	44

## 1. INTRODUCTION

---

The purpose of this manual is to give a complete and detailed description of the functionality and the characteristic of the communication facilities supported by the AC-20 controller.

The controller, in its maximum expansion is provided with the following communication lines:

- **Serial line for controller programming.**

It is a standard RS232 line, easily interfaceable through a plug-in connector with Personal Computers and programming terminals and it is used for downloading, off-line, configurations and start up parameters into the controller memory.

Furthermore, this line can be used, after the downloading operations, to debug the control strategy, by real time monitoring the internal variables. In this case, for implementing this debugging facility, extremely useful during the start up of the controller and the check of the strategy, a special software, to be installed on the PC, is required.

The line baud rate and configuration is fixed to :9600,E,7,2 (9600 baud, even parity, 7 bit data and 2 stop bits) and cannot be changed by the user.

- **Serial line to the supervisory system.**

This RS485 is dedicated to the communication with a supervisory system. It uses the JBUS protocol, Master - Slave type, with a subset of the standard JBUS functions, necessary for implementing the supervisory operation.

The controller has the Slave status on the JBUS protocol; this means that the controller is continuously analyzing the data flow on the line to detect requests and commands directed to itself and, when detected, transmitting back the responses to the supervisory system. It never transmit any data unless for responding to request from the supervisory system.

- **Serial line to the auxiliary unit.**

This RS485 line is dedicated to the communication between the remote I/O unit and the controller. It is a Master Slave connection, where the controller is the Master with a proprietary protocol. The information about this protocol are not supposed to be released to the users because the communication with the remote unit is automatically implemented by the hardware and the firmware of the instruments, already.

- **Local network for peer-to-peer communication.**

This is an high speed network (2,5 Mbit/Sec) capable to implement a peer to peer and broadcast communication between all the equipment's (both from ASCON and 3rd party) connected to the network.

One of most common application of this network is the softwiring between the AC controllers, for real time interconnecting of points of control strategies residing on different controllers, in order to implement a cluster wide control strategy

## 2. MODBUS & JBUS

### 2.1 Differences between MODBUS and JBUS

The only difference between the two protocols relates to the value of the address field of a message. The address values in the tables below are correct for the JBUS, while for MODBUS they must be decremented by 1. For instance, if a variable is listed at address 1, the correct address field for JBUS is equal 1, while for MODBUS is equal 0.

Furthermore, the lower value of an address is 0 for MODBUS and 1 for JBUS.

### 2.2 References

<b>GOULD</b>	Gould Modbus Protocol Reference Guide (PI-MBUS-300 Rev. B)
<b>APRIL</b>	JBUS Specification
<b>GLOBAL ENG. DOC.</b>	EIA STANDARD RS -485

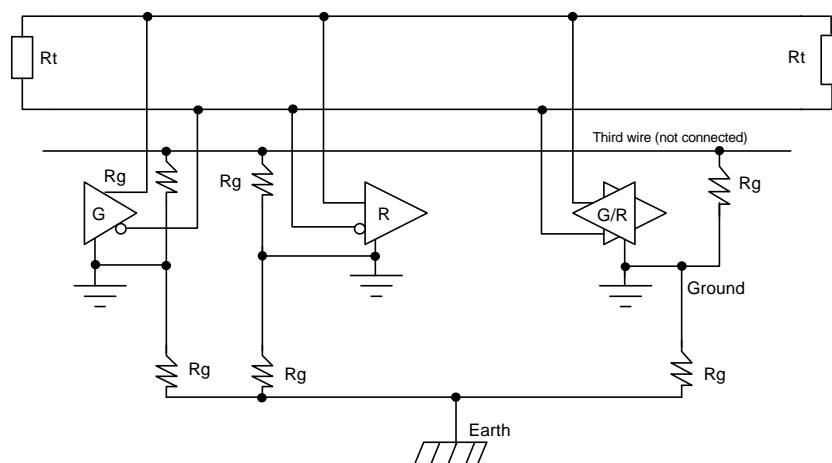
### 2.3 Hardware

The hardware must be compatible with the RS-485 standard.

Physically, the transmission line consists of a twisted pair cable with 120  $\Omega$  characteristic impedance. There are 3 switches (S1, S3, S4) on the controller CPU card, selecting the following functionality's.

Switch	Function	Function
	S1-A = No termination	S1-B = 120 $\Omega$ termination
	S3-A = Pull-down resistance	S3-B = No resistance
	S4-A = No resistance	S4-B = Pull-up resistance

The default positions of the switches, when the instrument leaves the factory, are in shaded gray. The termination (S1-B) is strongly recommended on the two controllers located at the end of the line, as shown in the drawing below, that has been copied from the original RS485 Standard specification.



- G Transmitter
- R Receiver
- G/R Bi-directional (Receiver/Transmitter) buffer
- Rt Termination resistance: the driving capability of the transmitter is minimum 32 receivers and 2 120  $\Omega$  resistors.
- Rg 100  $\Omega$  resistors.

## 2.4 The MODBUS Protocol

---

The MODBUS protocol specifies the structure of the communication, defining all the possible transactions that can occur on the communication line. A communication transaction occurs between the "Master" device, that is responsible of initiating the communication activity, sending to the "Slave" a message specifying the operation requested, and one or more "Slaves" devices that respond to the commands sent by the Master. In details, the protocol defines procedures and messages format, during the transaction between the Master and the Slaves, the mechanism of identification of any device on the line and the way errors are handled.

The protocol allows one Master and up to 247 Slaves on the communication line. This limit is several times greater than the 32 physical devices maximum, allowed by the RS485 standards on a single piece of cable, due to the driving capabilities of the transmitter ICs. This implies that the user can add more than 31 Slave device, inserting some buffer units on the line in order to respect the driving capabilities of the ICs. But he never can exceed the number of 247 Slave because of the logical limit of the protocol.

All the transactions are initiated by the Master. Two types of transactions exists:

- the one directed to a single Slave, consisting of a request from the Master and a reply from the Slave,  
and
- the broadcast one consisting of a message from the Master, directed to all the Slaves, without any reply from them.

Some of the characteristics of the protocol are not fixed and can be selected by the user. These are: interface standards, baud rate, parity, number of stop bits. The protocol provides two communication "modes", the ASCII and RTU (binary). Only the more efficient RTU mode is available on the ASCON controllers because the ASCII one is not used in high demanding applications. The JBUS protocol is perfectly identical to MODBUS with the only exception, mentioned previously, of the addresses values conventions. With MODBUS, the initial address is zero (0000 = 1st address) while with JBUS, the initial address is one (0001 = 1st address). A parameter that in Modbus is provided at a certain address, in JBUS is found at an address value greater of one.

Therefore, unless explicitly mentioned, all the information's below, even if referenced to MODBUS only, are valid for both the protocols.

## 2.5 Message Structure

---

The information's exchanged between two device are encapsulated in a packet, required for messages dispatch, data interpretation and data integrity protection. The packet is sent by the receiver of the source device, through the cable line, to the receiver of the destination. Each packet has a predefined format, specified in details by the MODBUS standard, and consists of the following fields, identical either for Master or the Slave.

- The Destination Address, that is the address of the device to which the packet is sent. If the address value is 0, the packet is transmitted to all the devices on the communication line (broadcast message).
- The Function Code, that specifies the type of operation that has been requested to perform.
- The Data, specifying the parameters or the results of the operation performed.
- The Error Check bytes , computed according to the CRC16 algorithm.

The receiving device checks always the integrity of the packet received, detecting any error in the packet structure, parity bit and CRC16 word. In case an error is detected, the message is considered invalid and rejected. If the message is a request sent to a Slave device, it doesn't take any action and doesn't send back any reply to the Master as in the situation where the packet is for another Slave device.

## 2.6 Address field

---

As mentioned below, each MODBUS transaction always involves the Master, governing the control of the line, and one Slave or, in the case of a broadcast packet, all the Slave. Only one transaction at a time can be carried on the line. The destination device is identified by the first byte of the packet, containing the address value. Each Slave has an unique address different from all the others. Valid address values are 1-247, in the case the packet has to be sent to an unique Slave, or 0 in the case of a broadcast operation, where the packet is sent to all the Slaves. The broadcast message is used to request operation that doesn't need a reply back to the Master, like variable setting and other assignment operation.

## 2.7 Function code

---

The second byte of the message identifies the function the Slave must perform. The Function Code received is transmitted back to the Master in the response of the Slave send at the end of the operation. On ASCON controller, this subset of the following MODBUS functions is provided:

- 01 Read Coil Status
- 02 Read Input Status
- 03 Read Holding Registers
- 04 Read Input registers
- 05 Force Single Coil
- 06 Preset Single register
- 07 Read Status ( Not implemented on AC Series)
- 15 Force Multiple Coils
- 16 Preset Multiple Registers

On ASCON controllers, functions 01 and 02 are functionally identical and interchangeable, as functions 03 and 04. For a full and detailed description of the functions, see chapter 3.

## 2.8 CRC16

---

The last two bytes of the packet represents a 16 bit word, named Cyclic Redundancy Check. This word is computed, according to the CRC16 algorithm, on the basis of the values of all the other bytes of the packet. The entire packet, consisting of Address, Function Code and Data fields, with the start, stop and parity bits stripped off, is considered as a single big binary number. The most significant bit (MSB) corresponds to the byte at the beginning of the packet, that is the one sent first. The message is first multiplied by  $x^{16}$  (shifted left by 16 bits) and then divided by  $x^{16}+x^{15}+x^2+1$ , expressed as a binary number 110000000000101. The integer quotient digits are ignored, while the 16 bit remainder (initialized at FFFFh at the start to avoid messages consisting exclusively of zeros) becomes the CRC16 code and it is appended at the end of the packet.

The receiving device check the integrity of the packet by dividing it by the same polynomial ( $x^{16}+x^{15}+x^2+1$ ), including the CRC, and verifying that the remainder is zero.

In most cases, the Universal Asynchronous Receiver and Transmitter chip (UART) computes directly the CRC but, because this components send the least significant bit (LSB) first instead of the MSB, the polynomial becomes reversed, in respect of the standard CRC calculation procedure. Furthermore, the MSB of the polynomial is dropped, since it affects only the quotient and not the remainder. This yields 10100000000000

The step by step procedure, that is used in real applications, for the CRC16 calculation is as follows:

- 1) Load a 16-bit register with FFFFh (all bits set to 1).
- 2) Execute the exclusive OR of the first byte with the top byte in the register and store the result in the register.
- 3) Shift the register to the right by one bit. The LSB bit goes out in the Flag bit.
- 4) If the Flag bit is 1, execute the exclusive OR of the polynomial 101000000000001 with the register and store the result in the register
- 5) Repeat the steps 3 and 4 eight times.
- 6) Execute the exclusive OR of the next byte with the top byte in the register and store the result in the register.
- 7) Repeat the steps 3 to 6 for all the bytes in the packed.
- 8) The content of the 16 bit register is the CRC code that is added to the packet.

### 2.8.1 How to compute CRC16 in Visual Basic

---

Function CRC16(String As String) As String

Dim N As Integer, i As Integer, NByte As Integer  
Dim CRC As Long, a As Byte  
Dim Buffer As String

NByte = Len(String)  
CRC = 65535

For i = 1 To NByte  
a = Asc(Mid\$(String, i, 1)) 'C(I)  
CRC = (CRC Xor a) And &HFFFF

For N = 0 To 7

If CRC And 1 Then  
CRC = (CRC \ 2)  
CRC = (CRC Xor 40961)  
Else  
CRC = CRC \ 2  
End If

Next

Next

Buffer = Right\$("0000" + Hex\$(CRC And &HFFFF), 4)  
CRC16 = Chr("&H" + Right\$(Buffer, 2)) + Chr("&H" + Left\$(Buffer, 2))

End Function

### 2.9 Messages Synchronization

---

A time pause of minimum 3.5 characters, is granted between the last byte of a packet and the first byte of the next one, in order to allow the Slave to safely identify the starting byte of the packet. The receiving device assumes that a byte received after a time corresponding to 3 characters is the first one of a new packet, that is the Address field.

### 3. MODBUS FUNCTIONS

---

This section provides a detailed description of the MODBUS functions implemented on ASCON controller of the AC series

#### 3.1 Read Output Status ( 01 )

---

This function allows the reading of the status (ON or OFF) of binary output variables. This functions is not allowed in broadcast mode (address = 0).

##### Request

The request consists of the Slave Address and the Function Code (01), followed by a 2 bytes Starting Address, specifying the first bit to read, and a 2 bytes Data Length, specifying the number of bits to read. The address values start from zero (bit 1 = 0) for the MODBUS and from 1 (bit 1 = 1) for JBUS.

Example: Read request of bit 0004 to 0015 status from Slave 17.

ADDR	FUNC	DATA start Addr HI	DATA start Addr LO	DATA bit # HI	DATA bit # LO	CRC HI	CRC LO
11	01	00	03	00	0C	CE	9F

##### Reply

The response consists of the Slave Address and the Function Code (01), followed by a single byte Data Count, specifying the number of data bytes returned, and the Data bytes reporting the status of the bits requested.

Each data byte reports the status of 8 bit, ordered with the least significant bit corresponding to the one at the Starting Address specified in the request and all the others ordered in the corresponding way. If the number of bits to read is not an exact multiple of 8, the remaining most significant bits of the last byte are filled with 0.

Example: Response to the request above.

ADDR	FUNC	DATA Byte Count	DATA bit 04..11	DATA bit 12..15	CRC HI	CRC LO
11	01	02	CD	0B	6D	68

#### 3.2 Read Input Status ( 02 )

---

This function allows the reading of binary input variables. It is completely identical to the previous function.

#### 3.3 Read Holding Registers ( 03 )

---

This function allows the reading the 16-bit (word) output registers, containing a numeric variables. This functions is not allowed in broadcast mode (address = 0).

##### Request

The request consists of the Slave Address and the Function Code (03), followed by a 2 bytes Starting Address, specifying the first register to read, and a 2 bytes Data Length, specifying the number of registers to read. The address values start from zero (word 1 = 0) for the MODBUS and from 1 (word 1 = 1) for JBUS.

Example: Read request of registers 069 to 071 from Slave 25.

ADDR	FUNC	DATA start Addr HI	DATA start Addr LO	DATA word # HI	DATA word # LO	CRC HI	CRC LO
19	03	00	44	00	03	46	06

**Reply**

The response consists of the Slave Address and the Function Code (03), followed by a single byte Data Count, specifying the number of data bytes returned, and the Data bytes with the registers content. Each register occupies two bytes, with the most significant one in the first byte.

Example: Response to the request above.

ADDR	FUNC	DATA Byte Count	DATA word 69 HI	DATA word 69 LO	DATA word 70 HI	DATA word 70 LO	DATA word 71 HI	DATA word 71 LO	CRC HI	CRC LO
19	03	06	02	2B	00	00	00	64	AF	7A

### 3.4 Read Input Registers ( 04 )

---

This function allows the reading of input registers. It is completely identical to the previous function.

### 3.5 Force Single Coil ( 05 )

---

This function allows the forcing of a single binary variable to ON or OFF state. This functions is not allowed in broadcast mode (address = 0).

**Request**

The request consists of the Slave Address and the Function Code (05), followed by a 2 bytes Starting Address, specifying the variable to set, a Data byte, specifying the binary value to set (ON = FFh or 255 while OFF = 00h), and an additional byte always set to 0. The address values start from zero (bit 1 = 0) for the MODBUS and from 1 (bit 1 = 1) for JBUS.

Example: Forcing request of bit 4 of Slave 47 to ON state.

ADDR	FUNC	DATA bit # HI	DATA bit # LO	DATA ON/OFF	DATA (zero)	CRC HI	CRC LO
2F	05	00	03	FF	00	7A	74

**Reply**

The response is identical to the message received and it's transmitted after the setting of the variable.

Example: Response to the request above

ADDR	FUNC	DATA bit # HI	DATA bit # LO	DATA ON/OFF	DATA (zero)	CRC HI	CRC LO
2F	05	00	03	FF	00	7A	74

### 3.6 Preset Single Register ( 06 )

---

This function allows the setting of a single 16 bit register. This functions is not allowed in broadcast mode (address = 0).

**Request**

The request consists of the Slave Address and the Function Code (06), followed by a 2 bytes Starting Address, specifying the variable to set, and a 2 byte

Data Word, specifying the value to set. The address values start from zero (word 1 = 0) for the MODBUS and from 1 (word 1 = 1) for JBUS.

Example: Set request of register 26 of Slave 38 to the value 926.

ADDR	FUNC	DATA bit # HI	DATA bit # LO	DATA WORD HI	DATA WORD LO	CRC HI	CRC LO
26	06	00	19	03	9E	DF	82

**Reply**

The response is identical to the message received and it's transmitted after the setting of the variable.

Example: Response to the request above.

ADDR	FUNC	DATA bit # HI	DATA bit # LO	DATA WORD HI	DATA WORD LO	CRC HI	CRC LO
26	06	00	19	03	9E	DF	82

### 3.7 Force Multiple Coils ( 15 )

This function allows the setting of the state of several binary variables located at contiguous addresses. This functions is not allowed in broadcast mode (address = 0).

**Request**

The request consists of the Slave Address and the Function Code (15), followed by a 2 bytes Starting Address, specifying the variable to set, a 2 byte Data Counter, specifying the number of bits to set, a Count byte, specifying the number of bytes containing the values to set, and the Data bytes with the status values.

Each Data byte reports the status of 8 bit, ordered with the least significant bit of the first byte corresponding to the Starting Address specified in the request and all the others ordered in a corresponding way . If the number of bits to set is not an exact multiple of 8, the remaining most significant bits of the last byte are filled with 0. The address values start from zero (bit 1 = 0) for the MODBUS and from 1 (bit 1 = 1) for JBUS.

Example: Force request of bits 1 to 4 on Slave 12. Bit 1 and 4 are forced to 1, the others to 0.

ADDR	FUNC	DATA start Addr HI	DATA start Addr LO	DATA bit # HI	DATA bit # LO	DATA Byte Count	DATA bit 1..4	CRC HI	CRC LO
0C	0F	00	00	00	04	01	09	3F	09

**Reply**

The response consists of the Slave Address and the Function Code, followed by the Starting Address and the number of bits written.

Example: Response to the request above

ADDR	FUNC	DATA start Addr HI	DATA start Addr LO	DATA bit # HI	DATA bit # LO	CRC HI	CRC LO
0C	0F	00	00	00	04	55	15

### 3.8 Preset Multiple Registers (16)

This function allows the setting of a block of 16 bit contiguous registers. This functions is not allowed in broadcast mode (address = 0).

**Response**

The request consists of the Slave Address and the Function Code (16), followed by a 2 bytes Starting Address, specifying the variable to set, a 2 byte Data Counter, specifying the number of registers to set, a Count byte, specifying the number of bytes containing the values to set, and the Data bytes with the values. The address values start from zero (bit 1 = 0) for the MODBUS and from 1 (bit 1 = 1) for JBUS.

Example: Set Request of the register at address 35 on the Slave 17 with the value 268

ADDR	FUNC	DATA start Addr HI	DATA start Addr LO	DATA word # HI	DATA word # LO	DATA Byte Count	DATA word 35 HI	DATA word 35 LO	CRC HI	CRC LO
11	10	00	22	00	01	02	01	0C	6C	87

**Reply**

The response consists of the Slave Address and the Function Code, followed by the Starting Address and the number of registers written.

Example: Response to the request above

ADDR	FUNC	DATA start Addr HI	DATA start Addr LO	DATA word # HI	DATA word # LO	CRC HI	CRC LO
11	10	00	22	00	01	A3	53

## 4. ERROR HANDLING

The MODBUS protocol specifies two types of errors, that must be handled in different ways: transmission errors and operating errors.

Transmission errors are due to an alteration of the structure, the character parity (if used) or the CRC16 of the packet received, because of problems of the line (noise, crosstalking, invalid connection) or the receiving/transmitting device (ICs failure or other misfunctionality's). Once the receiving device has detected such an error, it assumes that the entire packet is invalid and doesn't send back any reply.

Instead, an operating error is generated when the structure of the packet is correct but the function requested cannot be executed for different reasons. Once the Slave has detected this type of error it replies with an exception message, consisting of its the Address, the modified Function Code, the Error code and the CRC. The Function Code returned is modified by setting the most significant bit set to "1" to flag that the packet is an exception message.

Example: Read request of bit 1185 of the Slave 10:

ADDR	FUNC	DATA start Addr HI	DATA start Addr LO	DATA bit # HI	DATA bit # LO	CRC HI	CRC LO
<b>0A</b>	<b>01</b>	<b>04</b>	<b>A1</b>	<b>00</b>	<b>01</b>	<b>AC</b>	<b>63</b>

### Reply

Because bit 1185 doesn't exist, the Slave detects an operating error and reply with the error "02" (ILLEGAL DATA ADDRESS). Note that the Function Code returned 81h (129) is the same of the request but the MSB, that is set to 1.

Example: Reply to the request above.

ADDR	FUNC	DATA Except. Code	CRC HI	CRC LO
<b>0A</b>	<b>81</b>	<b>02</b>	<b>B0</b>	<b>53</b>

### 4.1 Exception code

Of the 8 exception codes specified in the MODBUS, the following 4 are the ones provided by the ASCON controller:

Code	Message Name	Description
<b>01</b>	ILLEGAL FUNCTION	The Function Code is illegal or not supported by the AC controller
<b>02</b>	ILLEGAL DATA ADDRESS	The Address referenced doesn't exist in the AC controller
<b>03</b>	ILLEGAL DATA VALUE	The Data value to set is out of the allowed ranges for the variable at the Address specified
<b>07</b>	NAK - NEGATIVE ACKNOWLEDGEMENT	The function cannot be performed due to the existing operating conditions or an attempt has been made to write at a read-only address.





### 5.3 The swap of the floating point words

---

The AC controllers, from the release 00I of AC10 and AC20 and release 00D of AC30 provide the functionality, on the Modbus protocol, of optionally swapping the order of the bytes, a floating point number consist of. The item **Sw. Float** is available in the AC “main Comm” menu, allowing two possible choices: Yes or No.

- **Sw. Float = No**

This is the standard choice, corresponding to the normal order of the bytes, as shown in the table below.

Floating point			
Word High		Word Low	
Byte 0	Byte 1	Byte 2	Byte 3

- **Sw. Float = Yes**

This choice performs the swap of all the bytes, as shown in the table below.

Floating point			
Word High		Word Low	
Byte 2	Byte 3	Byte 0	Byte 1

## 5.4 Logic variables ( Coils )

JBUS Address	Prograph Module	Variable	Write Enabled
1	DI #1	Logic input #1	NO
2	DI #2	Logic input #2	NO
3	DI #3	Logic input #3	NO
4	DI #4	Logic input #4	NO
5	DI #5	Logic input #5	NO
6	DI #6	Logic input #6	NO
7	DI #7	Logic input #7	NO
8	DI #8	Logic input #8	NO
9	DI #9	Logic input #9 auxiliary unit	NO
10	DI #10	Logic input #10 auxiliary unit	NO
11	DI #11	Logic input #11 auxiliary unit	NO
12	DI #12	Logic input #12 auxiliary unit	NO
13	DI #13	Logic input #13 auxiliary unit	NO
14	DI #14	Logic input #14 auxiliary unit	NO
15	DI #15	Logic input #15 auxiliary unit	NO
16	DI #16	Logic input #16 auxiliary unit	NO
17	DI #17	Logic input #17 auxiliary unit	NO
18	DI #18	Logic input #18 auxiliary unit	NO
19	DI #19	Logic input #19 auxiliary unit	NO
20	DI #20	Logic input #20 auxiliary unit	NO
21	DI #21	Logic input #21 auxiliary unit	NO
22	DI #22	Logic input #22 auxiliary unit	NO
23	DI #23	Logic input #23 auxiliary unit	NO
24	DI #24	Logic input #24 auxiliary unit	NO
25	DI #25	Logic input #25 auxiliary unit	NO
26	DI #26	Logic input #26 auxiliary unit	NO
27	DI #27	Logic input #27 auxiliary unit	NO
28	DI #28	Logic input #28 auxiliary unit	NO
29	DI #29	Logic input #29 auxiliary unit	NO
30	DI #30	Logic input #30 auxiliary unit	NO
31	DI #31	Logic input #31 auxiliary unit	NO
32	DI #32	Logic input #32 auxiliary unit	NO
33	DO #1	Logic output #1	NO
34	DO #2	Logic output #2	NO
35	DO #3	Logic output #3	NO
36	DO #4	Logic output #4	NO
37	DO #5	Logic output #5	NO
38	DO #6	Logic output #6	NO
39	DO #7	Logic output #7	NO
40	DO #8	Logic output #8	NO
41	DO #9	Logic output #9 auxiliary unit	NO
42	DO #10	Logic output #10 auxiliary unit	NO
43	DO #11	Logic output #11 auxiliary unit	NO
44	DO #12	Logic output #12 auxiliary unit	NO
45	DO #13	Logic output #13 auxiliary unit	NO
46	DO #14	Logic output #14 auxiliary unit	NO
47	DO #15	Logic output #15 auxiliary unit	NO
48	DO #16	Logic output #16 auxiliary unit	NO
49	DO #17	Logic output #17 auxiliary unit	NO
50	DO #18	Logic output #18 auxiliary unit	NO
51	DO #19	Logic output #19 auxiliary unit	NO
52	DO #20	Logic output #20 auxiliary unit	NO
53	DO #21	Logic output #21 auxiliary unit	NO
54	DO #22	Logic output #22 auxiliary unit	NO
55	DO #23	Logic output #23 auxiliary unit	NO
56	DO #24	Logic output #24 auxiliary unit	NO
57	DO #25	Logic output #25 auxiliary unit	NO
58	DO #26	Logic output #26 auxiliary unit	NO
59	DO #27	Logic output #27 auxiliary unit	NO
60	DO #28	Logic output #28 auxiliary unit	NO
61	DO #29	Logic output #29 auxiliary unit	NO
62	DO #30	Logic output #30 auxiliary unit	NO
63	DO #31	Logic output #31 auxiliary unit	NO
64	DO #32	Logic output #32 auxiliary unit	NO
65	MV #1 HCMV #1	DA/M Auto/man mode from keyb. #1 <sup>(1)</sup> (0 = Automatic, 1 = Manual)	YES

JBUS Address	Prograph Module	Variable	Write Enabled
66	MV #2	DA/M Auto/man mode from keyb. #2 <sup>(1)</sup> (0 = Automatic, 1 = Manual)	YES
67	MV #3 HCMV #2	DA/M Auto/man mode from keyb. #3 <sup>(1)</sup> (0 = Automatic, 1 = Manual)	YES
68	MV #4	DA/M Auto/man mode from keyb. #4 <sup>(1)</sup> (0 = Automatic, 1 = Manual)	YES
69	MV #1	DCAM Auto/man mode from comp. #1 <sup>(2)</sup> (0 = Automatic, 1 = Manual)	YES
70	MV #2	DCAM Auto/man mode from comp. #2 <sup>(2)</sup> (0 = Automatic, 1 = Manual)	YES
71	MV #3	DCAM Auto/man mode from comp. #3 <sup>(2)</sup> (0 = Automatic, 1 = Manual)	YES
72	MV #4	DCAM Auto/man mode from comp. #4 <sup>(2)</sup> (0 = Automatic, 1 = Manual)	YES
73	MV #1 HCMV #1	Hold status output #1 (0 = Normal, 1 = Hold)	NO
74	MV #2	Hold status output #2 (0 = Normal, 1 = Hold)	NO
75	MV #3 HCMV #2	Hold status output #3 (0 = Normal, 1 = Hold)	NO
76	MV #4	Hold status output #4 (0 = Normal, 1 = Hold)	NO
77..84	CDIO #1	DIO1 ÷ DIO8 digital I/O from computer	YES
85..92	CDIO #2	DIO1 ÷ DIO8 digital I/O from computer	YES
93..100	CDIO #3	DIO1 ÷ DIO8 digital I/O from computer	YES
101..108	CDIO #4	DIO1 ÷ DIO8 digital I/O from computer	YES
109	PRG #1	Run the program at block PRG_#1	NO
110	PRG #1	Hold the program at block PRG_#1	NO
111	PRG #2	Run the program at block PRG_#2	NO
112	PRG #2	Hold the program at block PRG_#2	NO
113	PRG #3	Run the program at block PRG_#3	NO
114	PRG #3	Hold the program at block PRG_#3	NO
115	PRG #4	Run the program at block PRG_#4	NO
116	PRG #4	Hold the program at block PRG_#4	NO

**Note**

- 1) Coils 65 to 68 activate input "DA/M" of the functional module "MV", with the effect of repeating the A/M key located on the front panel of the controller.
- 2) Coils 69 to 72 modify the "DCAM" field of the functional module g the controller to operate in manual mode without outlining this condition on the controller displays. The coils 69 to 72 have a lower priority than coils 65 to 68 (0=Auto, 1=Manual).

## 5.5 Numeric variables ( Register )

### Supervisory Section

JBUS Address	Prograph Module	Type	Variable	Write Enabled
1	AI #1	FP	Input #1 in engineering units	NO
3	AI #2	FP	Input #2 in engineering units	NO
5	AI #3	FP	Input #3 in engineering units	NO
7	AI #4	FP	Input #4 in engineering units	NO
9	AI #5	FP	Input #5 in engineering units	NO
11	AI #6	FP	Input #6 in engineering units	NO
13	AI #7	FP	Input #7 in engineering units	NO
15	AI #8	FP	Input #8 in engineering units	NO
17	AI #1	FP	Pulse input in engineering units	NO
19	AO #1	FP	Output #1 in %	NO
21	AO #2	FP	Output #2 in %	NO
23	AO #3	FP	Output #3 in %	NO
25	AO #4	FP	Output #4 in %	NO
27	AO #5	FP	Output #1 aux. Unit AAC-EU/88/4	NO
29	AO #6	FP	Output #2 aux. Unit AAC-EU/88/4	NO
31	AO #7	FP	Output #3 aux. Unit AAC-EU/88/4	NO
33	AO #8	FP	Output #4 aux. Unit AAC-EU/88/4	NO
35	SDV #1	FP	Set point from computer #1	YES
37	SDV #2	FP	Set point from computer #2	YES
39	SDV #3	FP	Set point from computer #3	YES
41	SDV #4	FP	Set point from computer #4	YES
43	SDV #1	W	Set point mode from keyboard #1 <sup>(8)</sup>	NO
44	SDV #2	W	Set point mode from keyboard #2 <sup>(8)</sup>	NO
45	SDV #3	W	Set point mode from keyboard #3 <sup>(8)</sup>	NO
46	SDV #4	W	Set point mode from keyboard #4 <sup>(8)</sup>	NO
47	SDV #1	FP	Controlling set point #1	NO
49	SDV #1	FP	Target set point #1	NO
51	SDV #1	FP	Local set point #1	NO
53	SDV #1	FP	Ratio set point #1	NO
55	SDV #2	FP	Controlling set point #2	NO
57	SDV #2	FP	Target set point #2	NO
59	SDV #2	FP	Local set point #2	NO
61	SDV #2	FP	Ratio set point #2	NO
63	SDV #3	FP	Controlling set point #3	NO
65	SDV #3	FP	Target set point #3	NO
67	SDV #3	FP	Local set point #3	NO
69	SDV #3	FP	Ratio set point #3	NO
71	SDV #4	FP	Controlling set point #4	NO
73	SDV #4	FP	Target set point #4	NO
75	SDV #4	FP	Local set point #4	NO
77	SDV #4	FP	Ratio set point #4	NO
79	SDV #1	W	Set point mode from computer #1 <sup>(9)</sup>	YES
80	SDV #2	W	Set point mode from computer #2 <sup>(9)</sup>	YES
81	SDV #3	W	Set point mode from computer #3 <sup>(9)</sup>	YES
82	SDV #4	W	Set point mode from computer #4 <sup>(9)</sup>	YES
83 ... 149	-	-	Available for future extensions	-
150 ... 200	-	-	System variables Area, listed at page 25	-

## Parameter Section

JBUS Address	Type	Prograph Module	Variable	Write enabled
201	FP	AI #1	TFIL - Filter time constant	YES
203	FP	AI #2	TFIL - Filter time constant	YES
205	FP	AI #3	TFIL - Filter time constant	YES
207	FP	AI #4	TFIL - Filter time constant	YES
209	FP	AI #5	TFIL - Filter time constant	YES
211	FP	AI #6	TFIL - Filter time constant	YES
213	FP	AI #7	TFIL - Filter time constant	YES
215	FP	AI #8	TFIL - Filter time constant	YES
217...232	W	DI #1 ÷ DI #16	Delay time of digital inputs DIN 1 .. DIN 16	YES
233...248	W	DO#1÷DO#16	Delay time of digital outputs DOUT 1 .. DOUT 16	YES
249	FP	SDV #1	SLU - Slope for Setpoint increase	YES
251	FP	SDV #1	SLD - Slope for Setpoint decrease	YES
253	FP	SDV #1	BETA - Local Set point weight	YES
255	FP	SDV #2	SLU - Slope for Setpoint increase	YES
257	FP	SDV #2	SLD - Slope for Setpoint decrease	YES
259	FP	SDV #2	BETA - Local Set point weight	YES
261	FP	SDV #3	SLU - Slope for Setpoint increase	YES
263	FP	SDV #3	SLD - Slope for Setpoint decrease	YES
265	FP	SDV #3	BETA - Local Set point weight	YES
267	FP	SDV #4	SLU - Slope for Setpoint increase	YES
269	FP	SDV #4	SLD - Slope for Setpoint decrease	YES
271	FP	SDV #4	BETA - Local Set point weight	YES
273	FP	PID #1	PBND - Proportional band	YES
275	W	PID #1	INT - Integral time	YES
276	FP	PID #1	DERT - Derivative time	YES
278	W	PID #1	CPID - Three terms enable	YES
279	FP	PID #2	PBND - Proportional band	YES
281	W	PID #2	INT - Integral time	YES
282	FP	PID #2	DERT - Derivative time	YES
284	W	PID #2	CPID - Three terms enable	YES
285	FP	PID #3	PBND - Proportional band	YES
287	W	PID #3	INT - Integral time	YES
288	FP	PID #3	DERT - Derivative time	YES
290	W	PID #3	CPID - Three terms enable	YES
291	FP	PID #4	PBND - Proportional band	YES
293	W	PID #4	INT - Integral time	YES
294	FP	PID #4	DERT - Derivative time	YES
296	W	PID #4	CPID - Three terms enable	YES
297	FP	MV #1 HCMV #1	CTRK - Computer tracking input	YES
299	FP	MV #1 HCMV #1	YMIN - Minimum output	YES
301	FP	MV #1 HCMV #1	YMAX - Maximum output	YES
303	FP	MV #2	CTRK - Computer tracking input	YES
305	FP	MV #2	YMIN - Minimum output	YES
307	FP	MV #2	YMAX - Maximum output	YES
309	FP	MV #3 HCMV #2	CTRK - Computer tracking input	YES
311	FP	MV #3 HCMV #2	YMIN - Minimum output	YES
313	FP	MV #3 HCMV #2	YMAX - Maximum output	YES

JBUS Address	Type	Prograph Module	Variable	Write enabled
315	FP	MV #4	CTRK - Computer tracking enable	YES
317	FP	MV #4	YMIN - Minimum output	YES
319	FP	MV #4	YMAX - Maximum output	YES
321	FP	LMT #1	LLO - Low limit	YES
323	FP	LMT #1	LHI - High limit	YES
325	FP	LMT #2	LLO - Low limit	YES
327	FP	LMT #2	LHI - High limit	YES
329	FP	LMT #3	LLO - Low limit	YES
331	FP	LMT #3	LHI - High limit	YES
333	FP	LMT #4	LLO - Low limit	YES
335	FP	LMT #4	LHI - High limit	YES
337	FP	ALM #1	VMIN - Low alarm threshold	YES
339	FP	ALM #1	VMAX - High alarm threshold	YES
341	FP	ALM #1	HYS - Alarm hysteresis	YES
343	W	ALM #1	SLC - Alarm source select	YES
344	FP	ALM #2	VMIN - Low alarm threshold	YES
346	FP	ALM #2	VMAX - High alarm threshold	YES
348	FP	ALM #2	HYS - Alarm hysteresis	YES
350	W	ALM #2	SLC - Alarm source select	YES
351	FP	ALM #3	VMIN - Low alarm threshold	YES
353	FP	ALM #3	VMAX - High alarm threshold	YES
355	FP	ALM #3	HYS - Alarm hysteresis	YES
357	W	ALM #3	SLC - Alarm source select	YES
358	FP	ALM #4	VMIN - Low alarm threshold	YES
360	FP	ALM #4	VMAX - High alarm threshold	YES
362	FP	ALM #4	HYS - Alarm hysteresis	YES
364	W	ALM #4	SLC - Alarm source select	YES
365	FP	ALM #5	VMIN - Low alarm threshold	YES
367	FP	ALM #5	VMAX - High alarm threshold	YES
369	FP	ALM #5	HYS - Alarm hysteresis	YES
371	W	ALM #5	SLC - Alarm source select	YES
372	FP	ALM #6	VMIN - Low alarm threshold	YES
374	FP	ALM #6	VMAX - High alarm threshold	YES
376	FP	ALM #6	HYS - Alarm hysteresis	YES
378	W	ALM #6	SLC - Alarm source select	YES
379	FP	ALM #7	VMIN - Low alarm threshold	YES
381	FP	ALM #7	VMAX - High alarm threshold	YES
383	FP	ALM #7	HYS - Alarm hysteresis	YES
385	W	ALM #7	SLC - Alarm source select	YES
386	FP	ALM #8	VMIN - Low alarm threshold	YES
388	FP	ALM #8	VMAX - High alarm threshold	YES
390	FP	ALM #8	HYS - Alarm hysteresis	YES
392	W	ALM #8	SLC - Alarm source select	YES
393	FP	HCMV #1	DBND - Heat - cool dead band	YES
395	FP	HCMV #2	DBND - Heat - cool dead band	YES
397	FP	AOUT #1:	TFIL - Filter time constant	YES
399	FP	AOUT #2:	TFIL - Filter time constant	YES
401	FP	AOUT #3	TFIL - Filter time constant	YES
403	FP	AOUT #4	TFIL - Filter time constant	YES
405	FP	AOUT #5	TFIL - Filter time constant	YES
407	FP	AOUT #6	TFIL - Filter time constant	YES
409	FP	AOUT #7	TFIL - Filter time constant	YES
411	FP	AOUT #8	TFIL - Filter time constant	YES
413	FP	FI	TFIL - Filter time constant	YES

JBUS Address	Type	Prograph Module	Variable	Write enabled
415..429	FP	CAIO #1	I/O 1..8 Analog I/O from/to Computer	YES
431..445	FP	CAIO #2	I/O 1..8 Analog I/O from/to Computer	YES
447..461	FP	CAIO #3	I/O 1..8 Analog I/O from/to Computer	YES
463..477	FP	CAIO #4	I/O 1..8 Analog I/O from/to Computer	YES
479	W	SELP #1	ITEM - Computer setting of an item <sup>(1)</sup>	YES
480	W	SELP #2	ITEM - Computer setting of an item <sup>(1)</sup>	YES
481	FP	MV #1	Y% - Controlled variable	YES
483	FP	MV #2	Y% - Controlled variable	YES
485	FP	MV #3	Y% - Controlled variable	YES
487	FP	MV #4	Y% - Controlled variable	YES
489	FP	PRG #1	NPGM - N° of the program selected	YES
491	FP	PRG #1	NSGM - N° of the running segment	YES
493	FP	PRG #1	SRT - Remaining time of the running segment of block PRG_#1	YES
495	FP	PRG #1	TCYC - Total n° of cycles of the program selected on block PRG_#1	YES
497	FP	PRG #1	ACYC - N° of the current cycle of the program selected on PRG_#1	YES
499	FP	PRG #2	NPGM - N° of the program selected	YES
501	FP	PRG #2	NSGM - N° of the running segment	YES
503	FP	PRG #2	SRT - Remaining time of the running segment of block PRG_#2	YES
505	FP	PRG #2	TCYC - Total n° of cycles of the program selected on block PRG_#2	YES
507	FP	PRG #2	ACYC - N° of the current cycle of the program selected on PRG_#2	YES
509	FP	PRG #3	NPGM - N° of the program selected	YES
511	FP	PRG #3	NSGM - N° of the running segment	YES
513	FP	PRG #3	SRT - Remaining time of the running segment of block PRG_#3	YES
515	FP	PRG #3	TCYC - Total n° of cycles of the program selected on block PRG_#3	YES
517	FP	PRG #3	ACYC - N° of the current cycle of the program selected on PRG_#3	YES
519	FP	PRG #4	NPGM - N° of the program selected	YES
521	FP	PRG #4	NSGM - N° of the running segment	YES
523	FP	PRG #4	SRT - Remaining time of the running segment of block PRG_#4	YES
525	FP	PRG #4	TCYC - Total n° of cycles of the program selected on block PRG_#4	YES
527	FP	PRG #4	ACYC - N° of the current cycle of the program selected on PRG_#4	YES
529..536	W	RTC	Sunday #1 .. #8 <sup>(2)</sup>	YES
537..544	W	RTC	Monday #1 .. #8 <sup>(2)</sup>	YES
545..552	W	RTC	Tuesday #1 .. #8 <sup>(2)</sup>	YES
553..560	W	RTC	Wednesday #1 .. #8 <sup>(2)</sup>	YES
561..568	W	RTC	Thursday #1 .. #8 <sup>(2)</sup>	YES
569..576	W	RTC	Friday #1 .. #8 <sup>(2)</sup>	YES
577..584	W	RTC	Saturday #1 .. #8 <sup>(2)</sup>	YES
585..600	W	DI #17 ÷ #32	Delay time of digital inputs	YES
601..616	W	DO #17 ÷ #32	Delay time of digital inputs	YES
617	W	16SW#1	Computer commands	YES
618	W	16SW#2	Computer commands	YES
619	W	16SW#3	Computer commands	YES
620	W	16SW#4	Computer commands	YES
621..650	-	FREE	-	-

The JBUS addresses of alarm modules, whose number is greater than 8, and the ones of the limit modules, whose number is greater than 4, are not included in the tables above and they are listed in the file "project\_name.mba". This file is generated automatically by the AC-Prograph/AC-Edit program when downloading the strategy into the AC controller. These addresses are not fixed, like the ones listed in the table above, and can change at every download operation. It is strongly suggested to complete the strategy and fully test it, before using the address information, in order to avoid to work with address information not updated with the current strategy running in the AC controller. Use the information in the "project\_name.mba" file at the completion of the project. For more information, refer to Chapter 6.

JBUS Address	Type	Prograph Module	Variable	Write enabled
651	-	RESERVED	-	-
652..669	-	FREE	-	-
670	W	-	Command register <sup>(3)</sup>	YES
671	W	PRG#1÷PRG#4	Prg N° - N° of the program to edit <sup>(4)</sup>	YES
672	W	PRG#1÷PRG#4	Node N° -N° of the segment to edit <sup>(4)</sup>	YES
673	W	PRG#1÷PRG#4	Time Type - Time units selection (0 = h:m, 1 = m:s)	YES
674	W	PRG#1÷PRG#4	Fast - Acceleration factor( 1..360 )	YES
675	W	PRG#1÷PRG#4	Cycle - Program repetition ( 0..999 )	YES
676	W	PRG#1÷PRG#4	Restart - Restart after power fail( 0.5 )	YES
677	FP	PRG#1÷PRG#4	Dev1 – Power fail condition <sup>(7)</sup>	YES
679	FP	PRG#1÷PRG#4	Dev2 – Power fail condition <sup>(7)</sup>	YES
681	W	PRG#1÷PRG#4	Tim1 – Power fail condition	YES
682	W	PRG#1÷PRG#4	Tim2 – Power fail condition	YES
683	FP	PRG#1÷PRG#4	Lo range - Program PV low scale	YES
685	FP	PRG#1÷PRG#4	Hi range – Program PV max scale	YES
687	FP	PRG#1÷PRG#4	Final Setpoint – Target SP in a segm.	YES
689	FP	PRG#1÷PRG#4	Aux SP – Auxiliary SP	YES
691	W	PRG#1÷PRG#4	Exec Time - Segment time <sup>(5)</sup>	YES
692	FP	PRG#1÷PRG#4	Allowed Dev - Maximum allowed deviation	YES
694	W	PRG#1÷PRG#4	DO1 DO16 - Pattern of the logic output of a Segment	YES
695	W	PRG#1÷PRG#4	End Prg. - Last Segment (1 =TRUE; 0 = FALSE )	YES
696	W	PRG#1÷PRG#4	D.P. - Digital point position	YES
697	W	RTC	Clock - Year <sup>(6)</sup>	YES
698	W	RTC	Clock - Month <sup>(6)</sup>	YES
699	W	RTC	Clock - Day <sup>(6)</sup>	YES
700	W	RTC	Clock - Day of the week <sup>(6)</sup>	YES
701	W	RTC	Clock - Hour <sup>(6)</sup>	YES
702	W	RTC	Clock - Minute <sup>(6)</sup>	YES

**Note**

1) The computer selects the item, specifying the digital output pattern, as follows.

Required item ( MSB )	Operation ( LSB )
Item number 0..15	0 to release , <> 0 to select

2) Bit 15 of the word is set to 1 when the pulse must be generated. The time of pulse generation is contained by bit 0 to 14, indicating the number of minutes from midnight, that is = (minute + (Hour)\* 60)

3) Command register: A write at this address doesn't alter the controller memory areas, but it activates one of the following operations:

Function	Code value
Delete All Programs	-2
Delete Current Segment	-1
Read Data	0
Write Data	1
Insert Segment after	2
Delete Current Program	3
Add a New Program ( = )	4
Add New Segment ( = )	5

**( = ) to be used only for building up a new program**

One of the important operation listed above is the Read (code = 0) and the Write (code = 1) Data. This operation performs the transfer of data between the memory area in the controller and the **Transfer Buffer**, where the data from MODBUS are temporary stored. This buffer is used for a limited set of registers, located at addresses 670 to 702.

One of the purpose of this buffer is to simultaneously write more registers. The Read and Write Data commands are governing the transfer between the Transfer Buffer and the memory area. When a register is read or written on MODBUS, the content of the Transfer Register is accessed. To access the controller memory a further operation is required: a the Read or Write Data command must be issued.

Therefore, to define a Segment, first of all, all the relevant registers must be written, and secondarily, a Write Data command must be issued.

For instance, to assign FAST ( address = 674) the new value of 2, execute the following steps:

- 1) Execute the **PSReg(Ind, 674, 2)** (Preset Single Register) function, that set to 2 the register at address 674, with the effect of writing in the Transfer Buffer, not in the register.
- 2) Execute the **PSReg(Ind, 670, 1)** (Preset Single Register) to issue a Write Data command of the Transfer Buffer into the memory area.
- 4) Writing the Program Number (address = 671) and The Node Number (address = 672) has the effect of updating all the information related to the Program and the Segment in the Transfer Buffer, that therefore can be read on MODBUS without issuing a Read Data command in advance. If the Program or the Node number doesn't exist an exception code (code = 8) is generated.
- 5) The Time of a segment is expressed in the units selected by the register at address **673**, between the two choices of Hours:Minutes or Minutes:Seconds. The number written is the time value expressed in the lower one of the selected units, that is Minute for the first choice or Seconds for the latter one.  
For instance, 2 hours and 45 minutes = 2 \* 60 + 45 = 165 minutes
- 6) The clock is read by reading register at address 697 to 702, considering the following tips:  
The Year is a number from 0 to 99  
The Month is a number from 1 to 12 (January = 1; December = 12)  
The Day of the month is at address 699 ( range = 1..31), while the Day of the Week is at address 700 (1 = Sunday, 2= Monday, ...7 = Saturday)

To write the clock these steps must be followed:

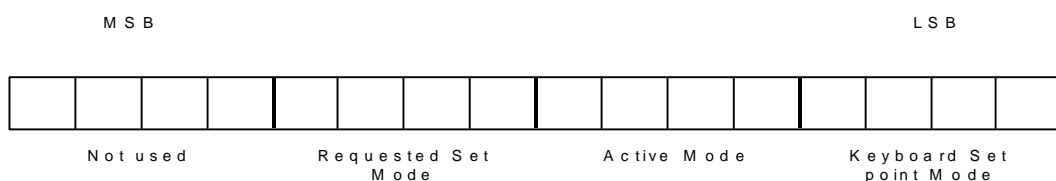
- a) Stop the clock sending Function 85 with the first Data byte set to 1.

Addr	85	01	01	CRC16	CRC16
------	----	----	----	-------	-------

- b) Write the new time and date at address 697 to 702. The content of these registers is set to the value of time and date when the clock is stopped: a tip to consider to decide which register to change during a setting operation
- c) Start the clock with Function 85 with the first Data byte set to 0.

Addr	85	00	00	CRC16	CRC16
------	----	----	----	-------	-------

- 7) To write a Float, it is not allowed to use 2 successive **PSReg** (Preset Single Register), but the **PMReg** (Preset Multiple Register) must be used
- 8) Operating mode form the Keyboard. It flags, in real time, the operating mode of the AC controller. The 16 bit word in the replay is structured with the following fields.



The coding of the Possible Set Mode field is as follows:

- 0 Local only
- 1 Remote only
- 2 Local / Remote
- 3 Local / Computer
- 4 Local / Remote / Computer

The coding of the Active Mode field is as follows:

- 0 None
- 1 Local
- 2 Remote
- 3 Computer

The coding of the Keyboard Setpoint Mode is as follows:

- 0 None
- 1 Local
- 2 Remote
- 3 Computer

9) Modification of the operating mode from computer

The 16 bit word, transmitted or received, has the following coding:

- 0 None
- 1 Local
- 2 Remote
- 3 Computer

## 5.6 Setting values and parameters of the Setpoint Programmer

---

The following step must be executed for entering a complete new program in the controller

- 1) If required, cancel all the programs, by issuing the command **PSReg(Ind, 670, -2)** (Preset Single Register). Write in register at address 671 the number of the new program with **PSReg(Ind, 671, NProg)**  
The Program Number has the following range: 0 to 15 for the first Programmer Block (PRG\_#1), 16 to 31 for the second one (PRG\_#2), 32 to 47 for the third one (PRG\_#3) and 48 to 64 for the fourth (PRG\_#4).
- 2) Write all the basic, segment independent, data of the program (Time Type, Fast, Cycle, Restart, Dev1, Dev2, Time1, Time2, Low Range, High Range). Write all the data related to Segment "0" or the initial Segment (Setpoint, Setpoint Aux, Delay, Max Deviation, Logic States, Format)
- 3) Issue the "Add a new Program" command with **PSReg(Ind, 670, 4)**
- 4) Write all the data related to Segment "N" (setpoint, setpoint aux, delay, max deviation, logic states, format)
- 5) Issue the "Add a new Segment" command with **PSReg(Ind, 670, 5)**
- 6) Repeat steps 5 to 6 for all the segments of the Program.

## 5.7 System variables

---

### Supervisory Section

JBUS Address	Type	Variable	Write Enabled
0151	W	Password ( 2 letters) (note 1 )	NO
0152	W	Password ( 2 letters) (note 1 )	NO
0153	W	Password ( 2 letters) (note 1 )	NO
0154	W	Group Enable (note 2 )	NO
0155	W	Product code (note 3 )	NO
0156	W	Product code (note 3 )	NO
0157	W	Manufacturer code ( ASCON = 600 )	NO
0158		Release code (note 3 )	NO
0159	-	Release code (note 3 )	NO
0191	FP	TS sampling time	YES

**Note**

- 1) The password consists of 3 words, each one containing 2 letters, in decimal format. It is organized as follows:

PASSWORD					
letter n°1	letter n°2	letter n°3	letter n°4	letter n°5	0
0151		0152		0153	

- 2) The Group enable allows the selection of the properties of 5 groups, each one using a 3 bits field. The MSB bit is not used. The Group are: A-B-C-Set Pnt-Manual

For the Groups A-B the choices are:

Visible and changeable = 10

Visible = 01

Not visible = 00

For the Group C-Set Pnt-Manual the choice are:

Changeable = 1

Unchangeable = 0

The organization of the fields in the word are as follows:

Group enable				
Manual	Set Pnt	C	B	A

- 3) The Product Code and the Release Code consist of 4 ASCII characters stored in 2 words.

Product Code			
letter n°1	letter n°2	letter n°3	letter n°4
0155		0156	

Release Code			
letter n°1	letter n°2	letter n°3	letter n°4
0158		0159	

Example : Release = " 00B"

32	48	48	66
$32 * 256 + 48 = 8240$		$48 * 256 + 66 = 12354$	

## 5.8 Reading and Writing the Configuration

The Configuration may be read or written by Modbus, using the functions

**Preset Multiple Register and Read Register**

These functions allow the reading and writing at all the locations of the controller database.

Register	Data Types	Read	Write
6000 ... 7FFF	NET	YES	NO
8000 ... 8FFF	MENU	YES	NO
9000 ... 9FFF	INTEGERS	YES	YES
A000 ... AFFF	REAL	YES	YES
B000 ... BFFF	STRINGS	YES	YES
C000... CFFF	COILS	YES	YES
D000..DFFF	REGISTERS	YES	YES
E000..E7FF	ARCNET producer variables	YES	YES <sup>(Note)</sup>
E800..EFFF	ARCNET consumer variables	YES	YES <sup>(Note)</sup>

**Note**

These variables can be written after a configuration download operation, only. The entire configuration file can be downloaded either through the RS232 port on the instrument front panel or through the "Main Comm." RS485 on the backplane.

This download of the configuration is an uninterruptible operation, that once started must be completed. The start and the conclusion of the download is flagged by the following predefined sequence events:

- 1) At the beginning the following operation must be executed:  
**PSReg(Address, 651, 0); (\* atomic operation beginning \*)**
- 2) After, the configuration is downloaded.
- 3) At the conclusion, the following operation must be executed:  
**PSReg(Address, 651, 1); (\* atomic operation end \*)**




---



---

All the register read and write to the area listed above, must be implemented through Preset Multiple Register function.

---



---

## 5.9 An example of configuring and programming through the RS 485

The following is an example of a Turbo Pascal program that reads an "\*.S19" file, generated by the AC\_PROGRAPH software and transfers it to the AC controller through the RS-485 ( Main Comm. ) JBUS link.

```

program config;
uses crt,mbuscom,setta;
var
bufin: array[1..16384] of Char; { 16K buffer}
infile :TEXT;
base:word;
riga,filename:string;
indirizzo:byte;
debugmode,fstring:boolean;
scelta:arofindex;

procedure Error(E: integer);
begin
case E of
1 : WriteLn('Error <1>: Illegal Function. ');
2 : WriteLn('Error <2>: Illegal Data Address. ');
3 : WriteLn('Error <3>: Illegal Data Value. ');
7 : WriteLn('Error <7>: NAK - Negative Acknowledgment. ');
16: WriteLn('Error !!!: No Answer. ');
17: WriteLn('Error !!!: Protocol Error. ');
18: WriteLn('Error !!!: Frame Error. ');
19: WriteLn('Error !!!: CRC Error. ');
end;
end;

function ByteToHex(B: byte): string;
const H: array [0..15] of char = ('0', '1', '2', '3', '4', '5', '6', '7',
'8', '9', 'A', 'B', 'C', 'D', 'E', 'F');
begin
ByteToHex := H[B div 16] + H[B and 15];
end;

procedure DispHex;
var I: integer;
begin
Write(#13, ' TX: ');
for I := 0 to TXNum do
begin
if (I mod 26) = 0 then
Write(#13, #10, ' ');
Write(' ', ByteToHex(TXSave[I]));
end;
end;

```

```
WriteLn;
Write(' RX:');
if RXNum < 0 then
  Write(#13, #10, ' <no answer>')
else
  for I := 0 to RXNum do
    begin
      if (I mod 26) = 0 then
        Write(#13, #10, ' ');
      Write(' ', ByteToHex(RXSave[I]));
    end;
  WriteLn;
end;

function GetAddress(w:word):word;
var t:word;

begin
  if w<$1000 then t:=$6000+w>(*netlist*)
  else
  if (w>=$1000) and (w<$2000) then t:=$A000+2*(w-$1000) (*reals*)
  else
  if (w>=$2000) and (w<$3000) then t:=$9000+(w-$2000) (*integers*)
  else
  if (w>=$3000) and (w<$4000) then begin t:=w-$3000;fstring:=TRUE;end>(*strings*)
  GetAddress:=t;
end;

function HexToInt(s:string):word;
const Hex1:set of char=['0'..'9'];
      Hex2:set of char=['A'..'F'];
      Hex3:array[1..4] of word=($1000,$100,$10,1);
var t,i:word;
    rt:array[0..10] of word;
begin
  t:=0;
  for i:=1 to 4 do
    begin
      if (s[i] in Hex1) then rt[i]:=ord(s[i])-48
      else rt[i]:=ord(s[i])-55;
      t:=t+Hex3[i]*rt[i];
    end;
  HexToInt:=t;
end;

procedure SendData(N:word);
var
  nb:string;
  w,k,j,i:word;
  flag:boolean;
  h:char;
  e:byte;
begin
  e:=100;

  while (e>0) do begin
    flag:=fstring;
    j:=N-3;i:=1;w:=0;
    while (i<2*j) do begin

      if ((flag) and (base and 1=1)) then begin (*odd string*)
        ModBusData.Data[w]:=0; (*starts with the terminator of the previous string*)
        flag:=FALSE;
        w:=w+1;
        base:=(base*15-1) div 2 +$B000>(*compute address of the first word*)
        end;

      if (flag) and ((base and 1)=0) then begin (*even string*)
        base:=(base*15) div 2+$B000>(*compute address of the first word*)
        flag:=FALSE;
        end;
    end;
  end;
```

```

nb:='00'+copy(riga,8+i,2);
k:=HexToInt(nb);
i:=i+2;
nb:='00'+copy(riga,8+i,2);
k:=k*256+HexToInt(nb);
ModBusData.Data[w] := Hi(k);
ModBusData.Data[w+1] := Lo(k);
i:=i+2;
w:=w+2;
end;
w:=(j div 2) ;
if fstring then inc(w);
PresetMRegs(Indirizzo,base,w);
E:=ReceiveAnswer;
if E <> 0 then Error(E);
DispHex;
if debugmode then repeat until keypressed;

    if (keypressed) then begin
        h:=readkey;
        if ord(h)=27 then begin ComClose;halt(1);end;
        end;

end;
end;

procedure openfile;
var i:integer;

begin
clrscr;
write('input filename ');
readln(filename);
assign(infile,filename);
settextbuf(infile,bufin);
{$i-}
reset(infile);
{$i+}
i:=ioresult;

case i of
    2: begin writeln('not existing file ');halt(1);ComClose;end;
    152:begin writeln('disk not ready');halt(1);ComClose;end;
    end;
end;

procedure ReadData;
var identifier,ad,nb:string[8];
flag:boolean;
N,data_ad:word;
i:integer;
begin
while not eof(infile)
do begin
readln(infile,riga);writeln(riga);
identifier:=copy(riga,1,2);
ad:=copy(riga,5,4);(*address*);
nb:='00'+copy(riga,3,2);
data_ad:=HexToInt(ad);

N:=HexToInt(nb);
flag:=FALSE;(*the default case *)
fstring:=FALSE;
case ord(identifier[2]) of
    ord('0'):flag:=FALSE;(*the beginning*)

    ord('1'):begin base:= GetAddress(data_ad);
        flag:=TRUE;
        end;(*trap here : NET,REAL,INT*)

    ord('2'):begin base:=$8000+data_ad;
        flag:=TRUE;
        end; (*menu*)

    ord('3'):flag:=FALSE;(*system data*)

```

```
ord('4'):begin base:=$C000+data_ad;
        flag:=TRUE;
        end;(*coils*)

ord('5'):begin base:=$D000+data_ad;
        flag:=TRUE;
        end;(*registers & double registers*)

ord('9'):flag:=FALSE; (*end of MOTOROLA like file*)

ord('A'):begin base:=$E000+data_ad;
        flag:=TRUE;
        end;(*ARCNET consumer list*)

ord('B'):begin base:=$E800+data_ad;
        flag:=TRUE;
        end;(*ARCNET producer list*)

end;(*end of case*)

if flag then SendData(N);
end;
end;


procedure sendmsg;
var e:byte;
begin
e:=100;
while (e>0) do begin
PresetReg(Indirizzo,651,0); (* atomic operation beginning *)
delay(100);
e:=ReceiveAnswer;
if E <> 0 then Error(E);
DispHex;

end;

ReadData;
e:=100;
while (e>0) do begin
PresetReg(Indirizzo,651,1); (*atomic operation ending *)
delay(100);
e:=ReceiveAnswer;
if E <> 0 then Error(E);
DispHex;
end;
end;

begin
clrscr;
setting(scelta);
debugmode:=FALSE;
if paramcount > 0 then debugmode:=TRUE;
if ComOpen(scelta[4]-1,scelta[1]-1,scelta[3]-1,scelta[2])
then begin
indirizzo:=1;
openfile;
sendmsg;
close(infile);
end
else writeln('No Communication Port Found ');
ComClose;
end.
```

**WARNING**



**Note:**  
 The addresses in the table above are intended for **16 bits word**. Therefore, the starting address of each register must be computed according to its size, adding to the starting address of its area all the words occupied by the previous registers.

For instance, REAL registers have a size of 2 words; therefore, if you want to read the real n°14, its address will be at  $A000 + 1C = A01C$ .

The STRINGS are using 15 bytes. Therefore string n° 8 is located at address  $B000 + 3C = B03C$ , that is an offset of 120 bytes or 60 word from the beginning of the STRING area.

If the string number is odd, also the bytes offset is odd. The starting address of the string, in words, is calculated by dividing the byte offset by 2. Therefore, the first register of the string is shared with the previous string, whose last character is in the lower byte of the register. For instance, string n° 3, has an offset of 45 byte and its starting register is at address **B02D**

**Reading the controller configuration**

A request to read the NET area must be issued.  
 This request is a Read Registers function (code n° 3) with a starting address at 0x6000 and the number of resister to read equal 50.

**TX:**  
 01 03 60 00 00 32 DA 1F

The controller replies as follows. Please, note that the data field can be different because they reflect the content of the configuration running.

**RX:**

01	03	64	00	09	10	0B	10	0C	20	07	10	0D	10	0E	20
08	10	0F	20	01	10	01	00	05	20	02	20	03	20	04	10
02	10	03	10	04	10	05	10	06	10	01	10	07	10	01	10
08	10	09	30	01	30	02	30	03	30	04	10	10	10	11	10
12	10	13	20	09	20	0A	10	14	10	15	20	05	10	0A	20
06	FF	FF	10	02	20	31	20	32	20	06	00	02	20	33	20
34	20	35	20	14	00	02	B0	98							

The first 3 bytes ( 01 03 64 ) are the Starting Address, the Function Code and the Byte Count, while the Netlist structure starts from 00 09 10 0B ... and ends with **FFFF always**.

Therefore, a technique to detect the end of the Netlist is to scan the data content and detect the FFFF terminator.

The configuration above is listed in a more comprehensive shape in the table below. This is obtained by listing the sequence of words starting from the first Data word in the reply. Each word reports the function block number or the reference of the variables connected to each input/output of the function block.

0009	Function n° 9 start = Frequency Input Block
100B	Frequency Low Limit
100C	Frequency High Limit
2007	Scale
100D	Range Low
100E	Range High
2008	Decimal Point
100F	Filter
2001	Digital Overrange
1001	Output
0005	Function n°5 start = 1 Bar Panel

To understand completely the Netlist, the following tip must be always remembered :

- The function number ranges from 1 to 100
- The Float variables range is 0x1000..0x1FFF with 1000 corresponding to FLOAT at address A000
- The Integer variables range is 0x2000..0x2FFF
- The String variables range is 0x3000..0x3FFF

Therefore, splitting the variable database of the controller in 3 vectors: INTEGERS, REAL and STRINGS, variable 100B corresponds to REAL[11], variable 2007 to INTEGER[7] etc..

All the function block have a predefined I/O variables structure, listed in the AC\_PROGRAPH manual, and one to one corresponding to the registers in the list of the Netlist above. Through the position in the list, a variable connected to an I/O of a Block can be easily identified and modified.

- For example, to modify the parameter Scale, that is a field with range 0..2, follow these steps
- 1) Identify Scale, the third parameter of the Function Block
  - 2) Read its address, that is : 2007
  - 3) Write the new value in the seventh register of the INTEGERS area. The INTEGERS area range is 0x9000..0x9FFF and, therefore, the location of register 7 is at 0x9007. The **Preset Multiple Registers** function that forces to 2 the register is:

<b>TX:</b>
01 10 90 07 00 01 02 00 02 B6 2F

<b>RX:</b>
01 10 90 07 00 01 9D 08

### 5.10 Menu reading

<b>TX:</b>
01 03 80 00 00 14 6C 05

<b>RX:</b>
01 03 28 00 09 30 05 00 00 00 05 30 06 00 0A FF
FF 00 05 00 22 00 02 30 1D 00 27 00 02 30 1E 00
2C 00 02 30 1F 00 31 00 19 30 20 A3 E6

As with the Netlist, the MENU composition is read by a Read Register Function at Starting Address 8000. The reply consists of a variable length list of DATA words, describing the MENUs, terminated by **FFFF**.

0009	Function 9
3005	String with its TAG
0000	Starting address (offset) of its instantiation in the NETLIST
0005	Function 5
3006	String with its TAG
000A	Starting address (offset) of its instantiation in the NETLIST

## 5.11 Real Reading

TX:							
01	03	A0	00	00	28	67	D4

RX:															
01	03	50	3B	20	3C	20	FF	E6	C9	A3	01	10	0B	10	3C
10	3D	10	3E	20	3D	20	3E	20	21	00	24	30	01	30	02
30	03	30	04	30	05	20	3F	10	04	10	07	10	02	10	3F
80	00	00	00	00	00	00	04	10	10	10	11	10	12	10	13
00	00	00	00	00	00	00	46	9C	40	00	00	00	00	00	46
9C	40	00	F4	A3											

The number of REAL, INTEGERS and STRINGS in a configuration is calculated by examining the Netlist and detecting the biggest address value for each type of variables.

For instance, by examining all the INTEGERS variables in the Netlist above, it appears that the biggest address value is 0x2030, that means there are 0x30 (48) integers.

## 5.12 Strings reading

TX:							
01	03	B0	00	00	64	62	E1

RX:															
01	03	C8	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	4F	46	46	20	4F	4E	4F	46	46	20	4F	4E	20	20
00	4F	46	46	20	4F	4E	41	49	31	41	49	32	20	20	00
41	49	33	41	49	34	41	49	35	20	20	20	20	20	00	50
56	31	20	20	20	20	20	20	20	20	20	20	20	00	4D	49
46	5F	31	20	20	20	20	20	00	20	20	20	00	31	4C	50
5F	31	20	20	20	20	20	00	FF	20	20	00	4F	46	46	20
4F	4E	41	49	31	41	49	32	20	20	00	41	49	33	41	49
34	41	49	35	20	20	20	20	00	50	56	31	20	20	20	20
20	20	20	20	20	20	20	20	00	4F	46	46	20	4F	4E	4F
46	46	20	4F	4E	20	20	00	4F	46	46	20	4F	4E	41	49
31	41	49	32	20	20	00	41	49	33	41	49	34	41	49	35
20	20	20	20	20	00	50	56	D8	20	20	C7	0A			

The strings consists of a total 15 bytes, of which, the first 14 are the **ASCII** characters of the strings, while the last one is the terminator, corresponding to '\0' in ANSI C. The C definition of this data structure is

```
typedef char tag[15]; /* 14 characters + terminator 0 */
```



## 6. Static and Dynamic ModBus / Jbus Addresses

---

As shown by the AC-Prograph / AC-Edit commands, not all the wires and function block parameters have a predefined ModBus / Jbus address. The ones listed by the commands are named static while the others are named dynamic.

### 6.1 Static address

---

The list of all the static addresses, supported by an AC controller, is shown in the tables of this manual. Through AC-Prograph / AC-Edit, the user can get a list of the static address of just the parameters related to his specific configuration, downloaded into the controller.

The big advantage of a static address is that its position never changes and it is completely independent by the specific configuration developed.

The disadvantage is that only a subset of parameters is supplied with static address, even if this set includes the most used ones.

### 6.2 Dynamic address

---

All the addresses of wires, function blocks, parameters of a control strategy are listed in the file "**project\_name.mba**", that is generated automatically by AC-Prograph / AC-Edit, in the directory where the strategy is stored, at the time of configuration download to the instrument.

Obviously, the address values change from a configuration to another one, because they are calculated at the time of download by AC-Prograph / AC-Edit. It is strongly suggested to read this file, by using a simple word processor (like Notepad in Windows), every time a configuration download is completed.

The dynamic address are located at address 30000 or higher. An example of the content of the "project\_name.mba" file is shown below:

```
"Module Type:EXP "
"Tag:Flow "
" Pin(1),      A :   40968 Double Register"
" Pin(2),      B :   40966 Double Register"
" Pin(3),      C :   40970 Double Register"
" Pin(4),      D :   40960 Double Register"
" Pin(5),      R :   40972 Double Register"

"C:\PROGRAMMI\AC-PROGRAPH\netimg   04-14-1998"

"Module Type:AI "
"Tag:Inlet "
" Parm(1),     INA:  36882 Register"
" Parm(2),     MS:   36884 Register"
" Parm(3),     TING: 36900 Register"
" Parm(4),     CLIN: 36901 Register"
" Parm(5),     TFIL: 41002 Double Register"
" Parm(6),     MLO:  41004 Double Register"
" Parm(7),     MHI:  41006 Double Register"
" etc....."
" etc....."
```



## 7. ARCNET

The ARCNET network is an high-speed token ring network (2.5 Mb/sec)

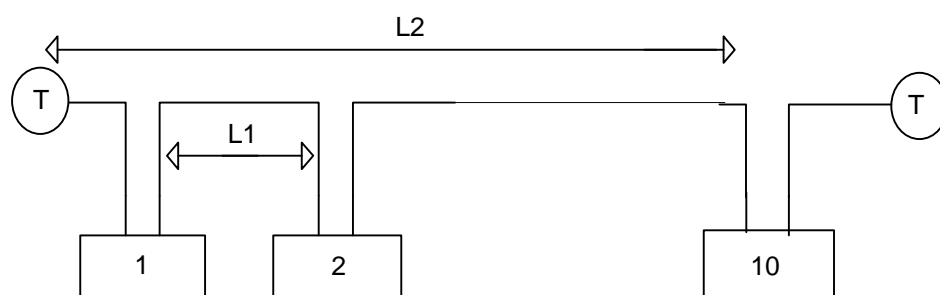
### References:

<b>SMC</b>	ARCNET Local Area Network Standard
<b>DataPoint</b>	ARCNET Designer's Handbook

### 7.1 Hardware

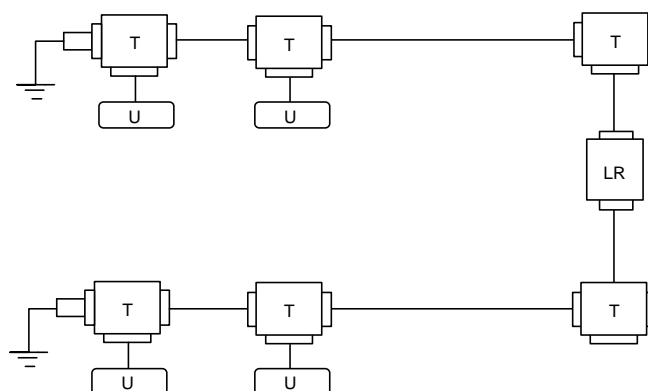
For the small local area networks, no additional hardware is required than the one directly provided by the controller.

A **maximum of 8 controllers** can be networked, by connecting with a proper cable the LAN+ e LAN- terminals of each controller in "Daisy Chain" or in a "backbone bus" configuration.



L1	Minimum distance between each controller	180 cm with twisted pair, 90 cm with coaxial cable
L2	Maximum total distance of the network	120 Mt. with twisted pair, 330 Mt. with coaxial
T	Termination resistance's	Same amount of the characteristic impedance

If more than 10 controllers need to be networked, the following architecture, based on the use of Repeaters, must be applied :



T	Transceiver, for the interconnection to the network
U	Device to connect to the network
LR	Local repeater

Parameter	COAX data	Twisted pair data
Wire type	RG62	Copper 22.24 or 26 AWG
DC resistance	1 ohm/100 metres	max. 8 ohm/100 metres
Characteristic impedance	93 ohm @ 1 MHz	105 ohm @ 1 MHz
Maximum attenuation	1.83 dB /100 metres @ 5 MHz	5.3 dB /100 metres @5MHz

## 7.2 ARCNET protocol for series AC controller

There are two types of protocol:

A) for communication between controllers

B) for communication between controllers and a computer

The structure of the data packet of the ARCNET protocol allows an easy identification of the type of protocol. In fact, the first byte of the packet flags the type of protocol.

Protocol type	Function type	Start Address Hi	Start Address Lo	Byte number
---------------	---------------	------------------	------------------	-------------

If the protocol type byte is zero, the protocol is of type A) otherwise it is of type B). The two protocols perform different tasks:

Protocol A) enables the users to share a common data base of variables (256 digital and 256 analog).

Protocol B) enables the users to read the configuration of each controller and to handle the entire data base, that is to access all the integer, floating point variables and strings).

## 7.3 Protocol type A

The protocol type A has the following structure. (each box is a byte)

0	0	1	0	2	1	2	0xFF	0xFF	0	200	0	100	0X40	0X48	0XF5	0XC3
---	---	---	---	---	---	---	------	------	---	-----	---	-----	------	------	------	------

In term of field, the organization is as follows

Protocol	Addresses Field						Control		Data Field							
Protocol Type	A0Hi	A0Lo	A1Hi	A1Lo	AnHi	AnLo	0xFF	0xFF	W0Hi	W0Lo	W1Hi	W1Lo	F00	F01	F02	F03

As shown in the picture above, the packet consists of the list of addresses of registers followed by the list of data values of these registers, ordered in the same sequence of the addresses.

The control word 0xFFFF separates the address from the data lists.

The 2 bytes address field indicates if a register is an Integer (address range 0..255) or a float (address range 256..511)

By applying these tips to the example above, the following consideration can be drawn:

- A0Hi,A0Lo is the address of Integer 0
- A1Hi,A1Lo is the address of Integer 2 **(1)**
- AnHi,AnLo is the address of Float 0 **(1)**
- 0xFF, 0xFF is the control separator
- W0Hi,W0Lo is the value of Integer 0
- W1Hi,W1Lo is the value of Integer 2
- F00,F01,F02,F03 is the value of the float F0

**(1):Please, note that the Address of an Integer have 0..255 range, while the float ones have 256..511 range.**

The data sharing architecture of this protocol sees as a **Producer** the controller that is generating the value of a shared variable, and **Consumer** all the others that need to know this value. Obviously, the status of Producer and Consumer of a controller is not fixed but is dynamically assigned according to the variables. In most situations, a controller is a Producer for some variable and Consumer for others.

The **Producer** controller has the job to update all the other with the value of its variables. Therefore, it builds up a packet like the one above and it broadcast it to all the other controllers on the network.

All the other controllers, when receiving the packet, become **Consumer** and scan the address list of the packet to check if there are variable they are concerned with, and, in affirmative case, they store the value of the variable in their memory. In this way all the Consumer have an update copy of the variables they require from the Producer.

## 7.4 Type B Protocol

In term of fields, the organization of type B packets, sent by a Computer asking for the values of some registers, are as follows.

Protocol type	Function type	Start Address Hi	Start Address Lo	RegNumberHi	RegNumberLo
---------------	---------------	------------------	------------------	-------------	-------------

The fields of the packet are:

Protocol Type	defining that this is a Type B packet
Function Type	indicating the data area to read, according to the table below.
Start Address	indicating the first address of the data area that the Computer wants to read.
Register Number	indicating the number of Registers, from the Starting Address must be read. The Register Number is limited by the maximum size of the packet, that is 256 bytes (128 words). Therefore, deducting the bytes used for the header information, the space useful for Data value is limited to apx. 240 bytes. This means that <b>RegNumber cannot be more than 120 for Integers and correspondingly for the other Data types.</b>

If The RegNumber of the request exceeds the limit an error code is generated.

The controller reply is organized as follows.

Protocol type	Function type	Start Address Hi	Start Address Lo	Data #0	Data # N-1	Data # N
---------------	---------------	------------------	------------------	---------	------------	----------

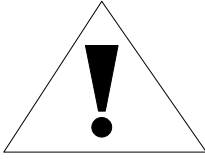
The Function Type field indicates the type of registers to read, according to the following table.

Function Type	Data Type	Function
0	NETLIST	Read
1	MENU	Read
2	REAL	Read
3	INTEGER	Read
4	STRING	Read
5	Network variables Producer	Read
6	Network variable Consumer	Read
15	Not Allowed	
16	Not Allowed	
17	REAL	Write
18	INTEGER	Write
19	STRING	Write

All the rules of calculating the addresses and the other tips about the structure of the data, presented in the Supervisory System chapter, related to MODBUS JBUS, apply to the ARCNET network. The only difference is that the starting address of the various data types are always 0 and not different for each type of data, like in MODBUS. The reason is that a separate field, the Function Type, is indicating the data type to access.

The following Warning gives some example of the address calculation procedure.

**WARNING**



**Note:**  
 The addresses in the table above are intended for **16 bits word**. Therefore, the starting address of each register must be computed according to its size, adding to the starting address of its area all the words occupied by the previous registers.

For instance, REAL registers have a size of 2 words; therefore, if you want to read the real n°14, its address will be at **1C**.

The STRINGS are using 15 bytes. Therefore string n° 8 is located at address **3C**, that is an offset of 120 bytes or 60 word from the beginning of the STRING area.

If the string number is odd, also the bytes offset is odd. The starting address of the string, in words, is calculated by dividing the byte offset by 2. Therefore, the first register of the string is shared with the previous string, whose last character is in the lower byte of the register. For instance, string n° 3, has an offset of 45 byte and its starting register is at address **2D**

A meaningful example is the packet to request the contents of a String register, precisely the **string n.3**, presented below.

Protocol Code	Function Code	Start Address Hi	Start Address Lo
1	4	0	16H

The reply to this request is the following, where the gray field is indicating the 15 characters string requested.

The first data byte, equal to 0, is the terminator of the previous string, the string n° 2. As illustrated in the Warning, all the strings with an odd number have, in the LSB of the first register, the last character of the previous string.

Pro	Fun	Hi	Lo																	
1	4	0	16	0	41	49	33	41	49	34	41	49	35	20	20	20	20	20	0	..

## 7.5 Error handling

As with the MODBUS protocol, ARCNET specifies two types of errors, that must be handled in different ways: transmission errors and operating errors.

Transmission errors are due to an alteration of the structure of the packet received, because of problems of the line (noise, crosstalking, invalid connection) or the receiving/transmitting device (ICs failure or other misfunctionality's). Once the receiving device has detected such an error, it assumes that the entire packet is invalid and doesn't send back any reply. The final result is that the Producer of the packet, after waiting hopeless the reply for a predefined amount of time (time-out), understands that its packet has gone lost and sends another request packet.

Instead, an operating error is generated when the structure of the packet is correct but the function requested cannot be executed for different reasons. Once the Controller has detected this type of error it replies with the following exception message, consisting of the Protocol Type and the Error code.

<b>Protocol Type</b>	Error Code = Function Code +128
----------------------	---------------------------------

## 8. AC Controller calibration through the serial port

The calibration of the instrument can be performed using the JBUS protocol on the RS485 port ( MAIN COMM port to the supervisory system).

The following table lists all the calibration operations that can be executed through the serial port. As shown, there are commands for inputs calibration, or single output calibration, clock check and display check.

Function code	Master Data	Slave data	Description
65	0	0	Calibration mode start
65	1	1	Calibration mode termination
66	X	CAL0	Low Range input calibration
67	X	CAL5	High Range input calibration
68	X	X	Calib out #1 L.R. (milliampere)
69	X	X	Calib out #1 H.R.(milliampere)
70	X	X	Calib out #2 L.R. (milliampere)
71	X	X	Calib out #2 H.R. (milliampere)
72	X	X	Calib out #3 L.R. (milliampere)
73	X	X	Calib out #3 H.R. (milliampere)
74	X	X	Calib out #4 L.R. (milliampere)
75	X	X	Calib out #4 H.R. (milliampere)
76	X	X	Calib out #1 L.R. (Volt)
77	X	X	Calib out #1 H.R. (Volt)
78	X	X	Calib out #2 L.R. (Volt)
79	X	X	Calib out #2 H.R. (Volt)
80	X	X	Calib out #3 L.R. (Volt)
81	X	X	Calib out #3 H.R. (Volt)
82	X	X	Calib out #4 L.R.. (Volt)
83	X	X	Calib out #4 H.R. (Volt)
84	X	X	LCD Test
85	1	1	Clock stop
85	0	0	Clock start
86	X	X	Reset the Controller

Function Code is the code of the operation to perform.

Master Data is the contents of the Data field when the request is sent to the controller.

Slave Data is the contents of the Data field when the reply is sent back from the controller. For instance, when the Input Calibration command is issued, the controller replies with the 16 bit value of the voltage measured on the input.

The following is an example of the Low range (zero) calibration operation.

The first operation to do is to start the Factory Test mode, using the command code 65

Request

01	41	00	00	51	CC
----	----	----	----	----	----

Reply

01	C1	00	71	90
----	----	----	----	----

To end the calibration, the command to exit from the Test mode must be sent.

Request

01	41	01	01	91	9C
----	----	----	----	----	----

Reply

01	C1	01	B0	50
----	----	----	----	----

The zero calibration command is sent, using Function Code 66.

Request

01	42	00	00	21	4C
----	----	----	----	----	----

Reply

01	C2	<b>00</b>	<b>8B</b>	E0	43
----	----	-----------	-----------	----	----

The data field content **008B** is the 16 bit value of the measured zero level.

Note

The second byte of the reply corresponds to the request code incremented by the hex value &H80. This is true with Factory Test mode, only.

## 8.1 Output calibration

---

The following is the command to calibrate output #4 in Volt , using Function Code 83.

Request

01	53	00	00	71	49
----	----	----	----	----	----

Once the controller has received this request, it forces the output to 95% of the full range, by setting the duty cycle of the PWM, controlling the output, to 95%.

At this point the user has to read the value of the output in Volt, using a multimeter. This measure must be written in location VOLT95[4] of the Controller in order to allow the controller to automatically compensate the calibration error.

If, for instance, the measure is 5,297 Volt, the user has to set VOLT95[4] = 5.297

This is achieved by writing the value 5.297 Volt, translated, as a IEEE floating point number, in 40A98106, in the 2 words at address 31 to 32, by using a JBUS Preset Multiple Registers (Function Code = 16). The addresses of all the calibration variables are listed in the table below.

This is the format of the Preset Multiple Register packet. The floating point value of the measure is in boldface.

01	10	00	1F	00	02	04	<b>40</b>	<b>A9</b>	<b>81</b>	<b>06</b>	17	51
----	----	----	----	----	----	----	-----------	-----------	-----------	-----------	----	----

The reply is the usual one of the Preset Multiple Function as outlined in the chapter 3.8

## 8.2 JBUS addresses in test mode

Address	Variable
1	Logic Input <sup>(4)</sup>
2	Logic output <sup>(5)</sup>
3	Output #1 <sup>(6)</sup>
4	Output #2 <sup>(6)</sup>
5	Output #3 <sup>(6)</sup>
6	Output #4 <sup>(6)</sup>
7	Output configuration <sup>(1)</sup>
8	Keyboard <sup>(2)</sup>
9	AI #1 <sup>(3)</sup>
10	AI #2
11	AI #3
12	AI #4
13	AI #5
14	AI #6
15	AI #7
16	AI #8
17	VOLT5[1] (MSW)
18	VOLT5[1] (LSW)
19	VOLT95[1] (MSW)
20	VOLT95[1] (LSW)
21	VOLT5[2] (MSW)
22	VOLT5[2] (LSW)
23	VOLT95[2] (MSW)
24	VOLT95[2] (LSW)
25	VOLT5[3] (MSW)
26	VOLT5[3] (LSW)
27	VOLT95[3] (MSW)
28	VOLT95[3] (LSW)
29	VOLT5[4] (MSW)
30	VOLT5[4] (LSW)
31	VOLT95[4] (MSW)
32	VOLT95[4] (LSW)
33	MAMP5[1] (MSW)
34	MAMP5[1] (LSW)
35	MAMP95[1] (MSW)
36	MAMP95[1] (LSW)
37	MAMP5[2] (MSW)
38	MAMP5[2] (LSW)
39	MAMP95[2] (MSW)
40	MAMP95[2] (LSW)
41	MAMP5[3] (MSW)
42	MAMP5[3] (LSW)
43	MAMP95[3] (MSW)
44	MAMP95[3] (LSW)
45	MAMP5[4] (MSW)
46	MAMP5[4] (LSW)
47	MAMP95[4] (MSW)
48	MAMP95[4] (LSW)
49	Low Range Input calibration value
50	High Range Input calibration value
51	LCD display temperature compensation <sup>(7)</sup>
52 ÷ 58	"Reserved"
59	Hide Ascon logo <sup>(8)</sup>

**Note:**

- 1) Output configuration. The word is split in four nibbles, describing the configuration of each output.

MSB

#4	#3	#2	#1
0..3	0..3	0..3	0..3

- 2) Keyboard. Only the LSB byte is used. Each bit is associated to a key on the instrument panel.

R/L	+	-	Disp	Page	AMan	<	>
-----	---	---	------	------	------	---	---

- 3) Analog Input. The unsigned binary number (0..65535) of the sampling value from the ADC (Analog to Digital Converter) is stored.

- 4, 5) Logic input/output. The LSB of the word gives the pattern of the 8 digital signals.

L8	L7	L6	L5	L4	L3	L2	L1
----	----	----	----	----	----	----	----

- 6) Analog Output

The value of the output, expressed as an unsigned binary number, with range 0..65535, is stored in this word.

For instance, to set the output, expressed in Volt with range 0..5, to 2,5 Volt, corresponding to 50% of the full scale, the binary number 32767 must be written. If the value to set is 3.12 Volt the value to write is  $40894 = (5/3.12) * 65536$

- 7) LCD display temperature compensation

Writing AAAA in this register, the controller will use a normal temperature compensation curve for its display. Otherwise, writing any other value than AAAA, the controller uses a different curve, suitable for an extended range.

- 8) ASCON logo Hide

By writing the value AA55 at this address, the ASCON logo is hidden. If a different value is written, the logo is shown. This functionality is available on AC20 and AC30, only.

### 8.3 Functions available

Not all the JBUS Functions are supported in Test mode. This is an updated list of all the functions, supported in the various operating conditions.

Function code	Description	Test Mode	Control Mode
01	READ COIL STATUS	NO	YES
02	READ INPUT STATUS	NO	YES
03	READ HOLDING REGISTERS	YES	YES
04	READ INPUT REGISTERS	YES	YES
05	FORCE SINGLE COIL	NO	YES
06	PRESET SINGLE REGISTER	YES	YES
15	FORCE MULTIPLE COILS	NO	YES
16	PRESET MULTIPLE REGISTERS	YES	YES

Test mode output

Request

01	41	01	00	50	5C
----	----	----	----	----	----

Reply

01	41	01	30	50
----	----	----	----	----