TENOR SERIES 100 INDUSTRIAL CONTROL COMPUTER

INSTRUCTION MANUAL

FEBRUARY, 1989

SERIES 100 INDUSTRIAL CONTROL COMPUTER SYSTEM

TABLE OF CONTENTS

Safety Notes

CAUTION

Please read the following notes regarding safe operation BEFORE proceeding to set up and operate the **Series 100 system**.

We strongly recommend the use of totally independent, non-semiconductor switches or hard-wired relay contacts for emergency stop purposes wherever personnel is exposed to machinery or a process under automatic control.

Wherever a system under processor control is likely to present a hazard during power failure, we recommend the **Series 100** be operable on 24VDC.

TENOR CONTROLS COMPANY draw the users attention to the following product warnings:

POWER SUPPLIES

Non-Operation

When not connected to a main supply, power units generate no output voltage. However, the electrolytic capacitors within the unit may contain a residual amount of energy. Care must always be taken before handling to insure these capacitors are fully discharged.

Operation

The processor **SERIES 100** is connected to the chassis ground point, therefore carefully select the supply ground when operating with a DC supply.

When connected to a power source, power supplies inherently have potentially dangerous conditions within. The user must insure that the ground connection on the processor is connected and the metal case is held at zero potential. Output voltages are normally isolated from the input voltage and from ground. It is the user's option whether to connect one terminal of the output to ground.

All electrical connections to the unit must be insulated for the voltage in use.

Poor electrical connections may result in damage to the unit or associated equipment.

When the unit is connected to a power supply, do not touch any of the components within the unit or the connections to the unit.

NICKEL-CADMIUM BATTERIES

Component materials in these batteries contain potassium hydroxide solution, nickel hydroxide and cadmium hydroxide. If the case is damaged and electrolyte comes into contact with the skin, rinsing in cold water is recommended.

ALUMINUM ELETROLYTIC CAPACITORS

The chemical constituents of the electrolyte vary according to manufacturer and type. If the case is damaged and electrolyte comes into contact with the skin, washing in cold water is recommended.

ALTHOUGH THE SYSTEM IS DESIGNED FOR MAINTAINING THE OPERATIONAL SAFETY OF THE PROGRAM IT IS POSSIBLE FOR SOLID STATE DEVICES TO FAIL IN AN UNSAFE CONDITION. SUCH A CONDITION COULD LEAD TO SUDDEN EQUIPMENT START UP, RESULTING IN OPERATOR INJURY OR EQUIPMENT DAMAGE.

TENOR CONTROLS COMPANY STRONGLY RECOMMEND THAT THE NEMA (NATIONAL ELECTRICAL MANUFACTURERS ASSOCIATION) ICS 3 - 304 PROGRAMMABLE CONTROLLER INSTALLATION GUIDELINES ARE FOLLOWED.

WARRANTY

ALL TENOR PRODUCTS SHIPPED AS SUCH, OR AS "COMPONENTS" OF "SYSTEMS" ARE WARRANTED AGAINST DFECTS IN MATERIAL AND WORKMANSHIP UNDER NORMAL RATED USE FOR APERIOD OF TWELVE MONTHS FROM THE DATE OF SHIPMENT. THIS EXPRESS WARRANTY IS IN LIEU OF AND EXCLUDES ALL OTHER REPRESENTATIONS MADE, BOTH EXPRESSED AND IMPLIED. **THERE ARE NO WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE FOR PRODUCTS COVERED BY THESE TERMS AND CONDITIONS**. TENOR WARRANTS THAT THE GOODS SOLD ARE AS DESCRIBED, BUT NO PROMISE, DESCRIPTION, AFFIRMATION OF FACT, SAMPLE, MODEL, REPRESENTATION, ORAL OR WRITTEN, SHALL BE A PART OF ANY ORDER, UNLESS SET FORTH IN THESE TERMS AND CONDITIONS, OR ARE IN WRITING AND SIGNED BY AN AUTHORIZED REPRESENTATIVE OF TENOR. IN NO EVENT SHALL TENOR BE LIABLE FOR ANY SPECIAL OR CONSEQUENTIAL DAMAGES.

TENOR "PRODUCT" WARRANTIES AND CONSULTATION AND ASSISTANCE OBLIGATIONS EXTEND ONLY TO "PRODUCT" AS EXPRESSLY DEFINED BY PURCHASE ORDERS, ASSOCIATED ENGINEERING DOCUMENTS AND APPROVALS. TENOR ASSUMES NO "SYSTEMS ENGINEERING" GUARANTEES OR RESPONSIBILITY IN ANY SENSE OF "SYSTEM" ABOVE AND BEYOND WHAT TENOR HAS SPECIFICALLY CONTRACTED TO FURNISH.

TENOR "PRODUCT" WARRANTIES AND CONSULTATION AND ASSISTANCE OBLIGATIONS EXTEND ONLY TO TENOR'S DIRECT CUSTOMER ONLY, NOT TO THIRD PARTIES.

ALL INFORMATION CONTAINED IN THIS MANUAL IS SUBJECT TO CHANGE WITHOUT NOTICE.

1.0 SERIES 100 INDUSTRIAL CONTROL COMPUTER SYSTEM

1.0 DESCRIPTION

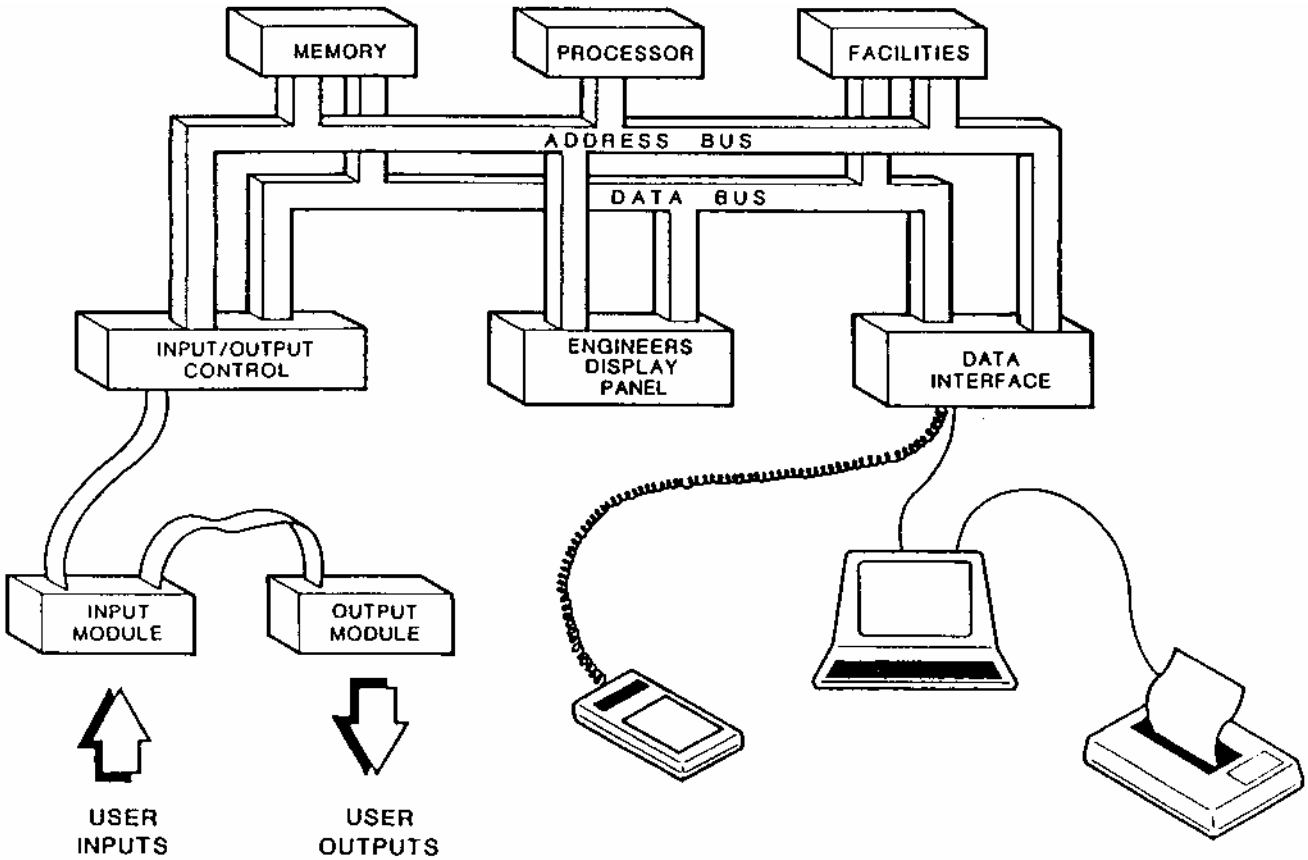
1.1 GENERAL

The **Series 100** Industrial Control Computer System is a family of Processors and Modules that enables automatic operation of monitoring and control functions. The **Series 100** family has been conceived and designed to interface with machinery commonly found in industry.

A series of ribbon cables connects various components of the system together. The **Series 100** uses this link to send data to and from I/O Modules. Input Module conditions and signals are converted into digital form for manipulation by the Processor as programmed by the User's Application program.

Similarly, the Processor sends the resultant digital information to an Output Module under program control. The Output Module converts the data into a form suitable for external use. To satisfy as many applications as possible, the **Series 100** Family offers a wide range of Input and Output Modules. Each is designed to fulfill a task appropriate to the particular conditioning need.

The choice of Modules required to satisfy any application depends on extent of Input Signal processing the desired form of an Output Signal. Once the proper modules have been selected, the system can be re-configured at any time by adding or changing Modules and re-writing user programs.



1.2 PROCESSORS

(photo of proc and mods.)

The Processors within the family all function in a similar fashion. Variations in memory type and capacity are available. The section decribing Processor lists this information in detail.

1.2.1 MEMORY

The two types of memory used in **Series 100** family are System and User Memories.

The area storing the Operating System instructions and functions are located EPROM. This information can only be read. The user is unable to alter it. It is often referred to as Firmware.

The Users programs are stored in the **Series 100** RAM and EPROM areas. Note that the User program uses EPROM that is not necessarily the same as that used by the Operating System. The Software developed for the application may be readily modified and updated.

1.2.2 AUXILLARY FUNCTIONS OF TIMERS, COUNTERS AND FLAGS

Timers, Counters and Flags are availaible in the Processor to note specific events.

Timer facilities are provided to control timing or permit a delay between events. The Counters provide a means to count up or down and permits set-point assignment. An event may be triggered when Set Point is reached. Flags enable events to be remembered. They may be thought of as a latching relay or flip-flop. Flags are updated by a user program.

1.2.3 I/O CONTROL

The I/O Control Section routes data and manages Input and Output Modules.

1.2.4 ENGINEERS DIPLAY PANEL

The Engineers Display Panel (EDP) is a display and a series of push buttons that provides system status information and allows the engineer to interrogate the operation of various parts of the system. It is located on the top face of the Processor.

1.3 INPUT/OUTPUT MODULES

The **Series 100** includes a broad range of digital, analog and numeric I/O Modules. These enable the **Series 100** to meet the most control system interface requirements.

Input and Output devices are connected to detachable 8 or 16 position Modules. These units are of heavy duty construction, UL recognized and have removable, polarized screw terminal connectors.

Eight point Modules have a green input LED and red output LED status indicators. An additional LED is used to indicate the presence of DC on the Module.

1.3.1 INPUT MODULES

Input Modules have up to 16 AC/DC Inputs per Module including units that interface to Thumbwheel Switches, Thermocouples and Pressure Transducers are available.

1.3.2 OUTPUT MODULES

The Output Modules provide AC and DC Switched Output functions in both relay and semiconductor form. Display Output, Analog Voltage or Current models are available.

All Modules include a 100-3-9080 cable and 100-3-9088 bracket unless noted in the Module specification.

2.0 PROCESSOR SPECIFICATIONS

2.1 GENERAL SPECIFICATIONS

There are two Processors in the Series 100 Product Line Family. They are:

100-7-9001-06 Processor with 32K User and 16K Variable Memory

100-7-9002-06 Processor, Dual Port, with 32K User and 16K Variable Memory

Processors are shipped with a 100-3-9080 Cable, 100-3-9086 End Link,100-3-9087 Cord, and (4) 100-3-9088 Brackets.

2.1.1 AC POWER SUPPLY

The **Series 100** has a voltage selector and an AC power input fuse located above the power socket, enabling the selection of 110 volts, 220 volts, or 240 volts; +10% ,-15% at 150VA max. A ground connection to the internal chassis is provided next to the socket.

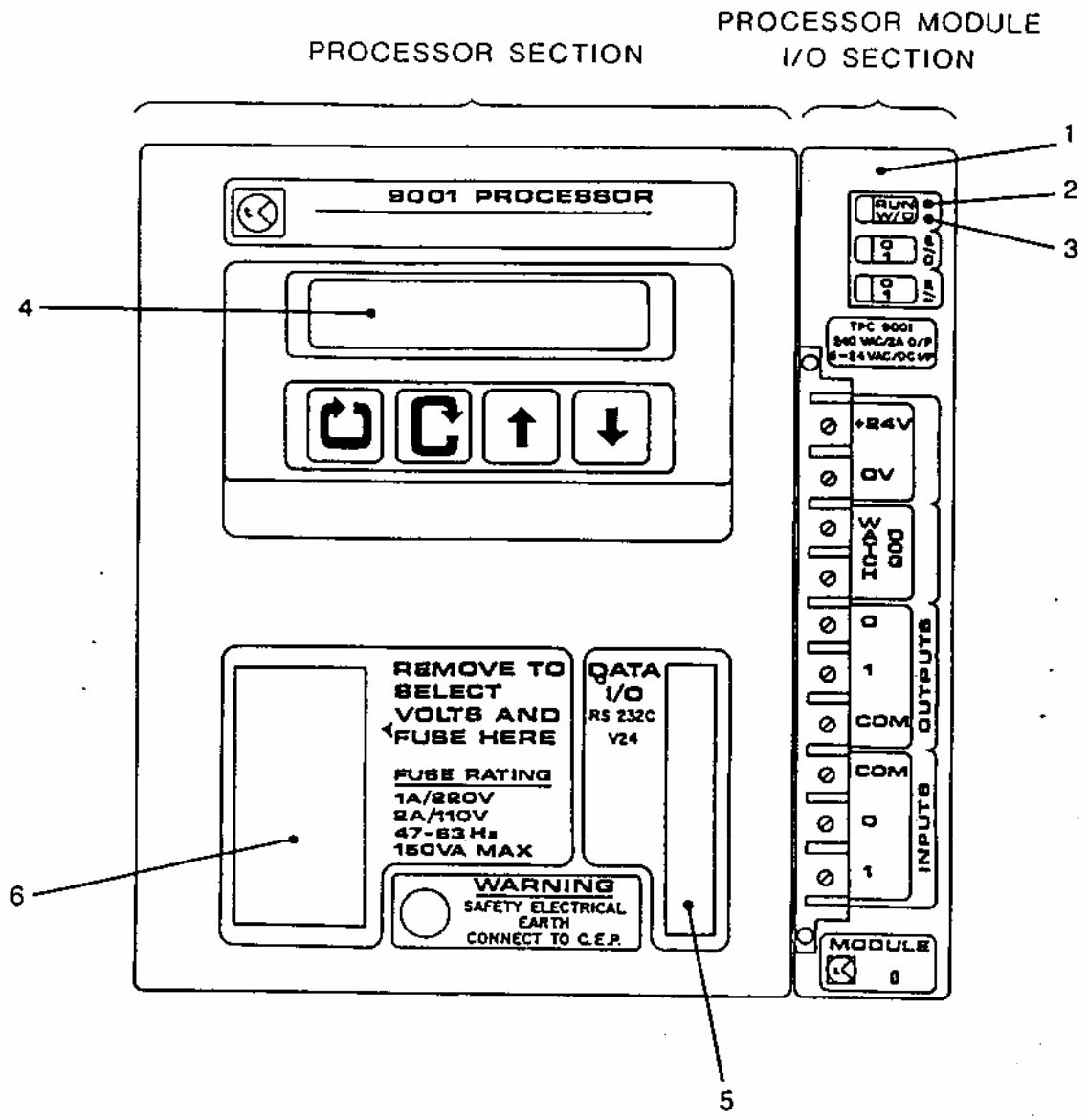
2.1.2 LOW VOLTAGE DC TERMINALS

Supply Input: fused 24VDC + 15%, -20% at 3A max
Supply Output: 24VDC q 20% 200ma max when using an AC power supply.

24 VDC at 200ma max can be used to drive a small I/O control system when the Processor is operating on AC power. It is also capable of supplying a trickle charge to a 24 volt battery system. The supply reverts to the DC mode without interrupting operation of the Processor if AC power fails.

2.1.3 POWER FAILURE DESCRIPTION

The processor will lock out if the supply drops below 75% of rated voltage for longer than 20 msec. Operation will be restored when the supply is stabilized for more than 1 second and can provide greater than 80% of its rated voltage.



1 POWER "ON" INDICATOR

2 PROCESSOR "RUN" INDICATOR

3 PROCESSOR W/D (WATCHDOG) INDICATOR

4 ENGINEERS DISPLAY

5 TERMINAL OR HAND-HELD PROGRAMMER CONNECTOR

6 MAINS CABLE CONNECTOR PLUS ACCESS POINT FOR FUSE AND MAINS SELECTOR (240, 110 Vac)

2.1.4 ENVIRONMENT

Operating temperature - 0 to 60 xC.

Storage temperature - -40 to 95 xC.

Humidity - 1% to 95% continuous, non-condensing.

Vibration - 10 to 55Hz, 1.5mm dual amplitude.

Shock - 10G, 11 msec

2.1.5 DATA RETENTION

All inputs, outputs, flags, variables, timer/counter set points and valuesare saved in battery supported CMOS RAM for at least 28 days when system is powered down. Loss of memory will cause Watchdog Lockout on subsequent power up.

2.1.6 PROGRAM MEMORY

This consists of battery supported CMOS, RAM or externally programmed EPROM.

2.1.7 INPUT/OUTPUT (I/O) CAPACITY

A maximum of 252 I/O points, in groups of 8 or 16, are available.

2.1.8 INTERNAL POWER SUPPLY

The Processor provides power to each of the Modules via the ribbon cables. The specification sheet on each Module includes a Supply Load requirement of "a" or "b" units. When the required Modules have been determined, the "a" and "b" units are totaled. If either the total "a" load exceeds 300 units or the total "b" load exceeds 200 units, then an additional Power Supply Module will be required. One "a" or "b" unit is equal to 10mA.

2.1.9 INTERNAL FLAGS (RELAYS)

The **Series 100** has 256 internal locations for storing results of logical operations. They do not need to interface directly with the I/O points. They are designated FO to F255. Flags designated as FO to F127 are always cleared on power up. Flags designated as F128 to F255 are retained at power down.

2.1.10 TIMERS/COUNTERS

Sixty-four timer/counters are available. Timer/counters 1-63 are individually programmable locations can be used as timers or counters. Timer delays ranging between 0.1 and 999.9 sec may be programmed in steps of 0.1 sec in the timer mode. Values up to 9999 may be accumulated in counter mode. Counter 0, is a High Speed counter, having a 10Khz clock rate.

2.1.11 PROCESSOR INPUTS X0.0, X0.1

Two opto-isolated inputs appear as filtered low voltage inputs when normally addressed. X0.0 is a high speed input to Counter 0. The speed of the input is 10KHZ.

Off Voltage - -27 to +1.2VDC

On Voltage - +4.5 to +27VDC

Input Current - 1.6mA typical at +5.0 VDC 20mA max

Turn On/Off - 12msec typ.

Isolation Voltage - 500VAC min.

2.1.12 PROCESSOR OUTPUTS Y0.0, Y0.1

Two Normally Open (NO) relay contacts, with overvoltage protection, are addressed as normal outputs.

Contact Rating - 2A at 240VAC/30VDC

Output Voltage - 264VAC max

Isolation Voltage - 1500VAC, 1minute

2.1.13 SPECIAL FUNCTION OUTPUTS

Processor Outputs, Y0.4 to Y0.7, are not available for connection to hardware. They have the following functions:

Y0.4 is energized from the start of user program execution at switch on until the first break.

Y0.5 can be used to inhibit the Force function by program as follows: During the normal I/O update, Y0.5 is transferred to an internal flag (called **PANEL**). If Y0.5 is set by program, then nothing can be changed through the Engineering Panel. **FORCE** registers can only be changed if Y0.5 is clear and **FORCE** flags allow a change.

Y0.5 may be set and cleared under program control; it is also cleared by entering **EDIT**, **CLEAR** command, **RUN** command, **LOAD** command, and **Watchdog.**

Y0.6 is energized for 10msec every 100msec.

Y0.7 is energized for 10msec every 1sec. Processor Inputs X0.2 and X0.3, are also not available for hardware connection, they have the following functions:

2.1.13 SPECIAL FUNCTION OUTPUTS CONTINUED

X0.2 is associated with the **PRINT** statement. The **PRINT** statement has been made **PART-COMPLETE** and the flag X0.2 is cleared. No other **PRINT** statements may commence until X0.2 is reset.

X0.3 is set at the start of **INPUT** and cleared on receipt of (CR). Note that it is essential to test X0.3 in a task other than one in which the **INPUT** statement is used. Because it is a **PART-COMPLETE** statement, it will hold up tasks, like waiting for an INPUT, until it has been executed.

X0.4, X0.5, X0.6, X0.7 are Engineer's Panel buttons which can override normal the display operation in programs. If buttons are numbered 1 to 4 from, left to right, the outputs are:

X0.4 = 3 (UP) **X0.6** = 1
X0.5 = 4 (DOWN) **X0.7** = 2

2.1.14 WATCHDOG OUTPUT

One NO contact with overvoltage protection closes when all hardware and software checks are correct.

Contact Rating - 2A at 240VAC/30VDC
Output Voltage - 264VAC max.
Isolation Voltage - 1500VAC for 1 minute.

2.1.15 SCAN TIME - 10msec/K of user memory

2.1.16 REAL TIME CLOCK

A battery supported clock circuit gives time of day, day of week and date with full leap year compensation. It may be set via pushbuttons on the Engineer's Panel.

2.1.17 FIRST SERIAL PORT

The RS232/RS422 interface allows the Processor to communicate with other devices using this standard. The RS232 interface provides full duplex communications. Data rates of between 300 and 4800 baud are available. Connections are made with a 25 pin D type connector.

Pin Connections:

Pin 1 & 7 - Ground, 0 volts
Pin 2 - TXD Transmit Data
Pin 3 - RXD Receive Data
Pin 4 - RTS Output
Pin 5 - CTS Input
Pin 10 - RS422 Data +
Pin 11 - RS422 Data -
Pin 13 - +8.0 Volts
Pin 14 - EDIT Enable
Pin 20 - DTR Output

2.1.17 FIRST SERIAL PORT CONTINUED

Application Programs may be written by connecting the editing cable to the RS232 Port on the Processor and to an ASCII VDU terminal. A Hand Held Programmer (HHP) may be substituted for the VDU terminal.

The port can be used to connect VDUs, printers, modems or serial communication devices to the **Series 100** system.

2.1.18 SECOND SERIAL PORT

The 100-3-9002 Processors have a second serial port. In addition to having the same capability as the first serial port, it provides access to the status of all input, output, flags, and numeric variables.

These are the available protocols:

1. Asynchronous Communications, using a human readable set of commands and responses.

2. Synchronous Communications, with multi-drop network capabilities for connection to data highways such as, TIway or PROway.

The interface is provided via 25 pin D type connector for RS232C or RS422. It may also use the RS423 standard for single point communication, and RS485 for multi-drop applications. All port points are fully isolated from Processor hardware.

These are the Data Type Address Ranges:

Inputs - X0.0 to X31.7

Outputs - YA0.0 to YH3.7

Flags - F0 to F127, Non-Retentative; F128 to F255, Retentative

Timer/Counter Set Point - SP0 - SP63

Timer/Counter Value Up - VU0 - VU63 note: Set Point and Value Up values 0 to 9999 are encoded as WI and WO

Variables: A(0,0,0) to A(84,84,84) to Z(84,84,84)

Inputs and Outputs have four possible states:

0 = OFF
1 = ON
2 = Forced OFF
3 = Forced ON

Flags may only assume states 0 or 1.

Interrogation requires an I/O, Flag or Variable address. The **Series 100** will reply with the status or value.

2.1.18 SECOND SERIAL PORT CONTINUED

Assignment requires an I/O, Flag, or Variable address followed by "=" and the desired ASCII character value.

Further Communications Protocol information is found in Appendix 2 of of the **Series 100** manual.

2.1.19 ELECTRICAL NOISE

Electric noise limited to NEMA ICS 2-230 standard.

2.1.20 CREEPAGE AND CLEARANCE

Limited to UL 508 and VDE 0110 ;Terminal breakdown - 3000V.

2.1.21 SIZE

191mm x 225mm x 180mm,
7.5" x8.9"x 7.1"

2.1.22 WEIGHT - 900 grams, 1.98 #, maximum

2.2 FEATURES

2.2.1 ENGINEERS PANEL

This is an 8 character, alpha-numeric LCD panel with four pushbuttons for review of the status of I/O, timer/counter, and clock data. The Engineer's Panel also allows adjustment of this information.

2.2.2 Power ON LED

When the AC or DC supply connected, and the Processor fuses are intact, the **Power ON LED** is lit.

2.2.3 RUN LED

This LED shows the operating state of the Processor. It flashes at 1 sec. intervals when the Processor is running an applications program. The LED is off when Processor is in Edit Mode or detected a fault and executed a failsafe.

2.2.4 POWER SUPPLY BOARD FUSE

The fuse used to protect the power supply board is located behind the I/O terminal block.

3.0 INSTALLATION

This section describes the procedures necessary for a proper installation of the **Series 100** Industrial Control Computer System.

SYSTEMS PLANNING
ASSEMBLING THE SYSTEM
TESTING THE PROCESSOR
MODULE INTERCONNECTION
I/O CONNECTIONS

The equipment should be unpacked and arranged such that all components are easily accessible. The Processor package should include a Processor, a Power Cable, two Mounting Brackets, a Ribbon Cable and an End Link. The I/O Modules are packed to include a Mounting Bracket and a Ribbon Cable.

3.1 SYSTEMS PLANNING

The following points should be considered during the planning process.

1. The interface has been designed to allow a full system with 32 I/O Modules may be distributed over a mimimum distance of 300 m (1,000 feet). Clock and data lines are buffered in each Module. For remote operation, 4 pair shielded cable, like Belden 28 AWG 9805 or 24 AWG 9831, is recommended. A voltage drop of 2.2V is allowed on 8.0V and 0V lines. Longer cables will require the use of 100-3-9070 Remote Power Supply, and 3 pair Belden 9805 cable.

2. Local Output Modules should occupy lower numbered locations and Input Modules higher numbered locations. This permits easy simulator insertion during test operations.

The Processor and Remote Power Supply provide power for each Module via ribbon cable. Each is able to drive a maximum of 16 Modules. The Processor supplies two different voltages known as 'a' and 'b' sources. When the number and type of Modules in a system has been determined, total the number of 'a' and 'b'. If the 'a' load exceeds 300 units, or the 'b' load exceeds 200 units, an additional 100-3-9070 Power Supply Module is required.

Supply load values for 'a' and 'b' are listed in the SPECIFICATIONS of each Module.

3. Keep high and low voltage signals separated.

4. Equipment mounted in an enclosure should should have an ambient temperature within 0 - 50 °C range.

5. I/O cables must be kept away from sources of electrical noise, such as switch-gears, inductive elements or large motors.

6. Series 100 equipment wired in the same phase as other switching machinery at the installation may be a source of electrical noise. This can be suppressed by adding in-line noise filters to the Processor power cable and Remote Power Supply Modules.

7. Transformers, inductors and other related equipment radiate electrical or magnetic noise. Since the **Series 100** monitors very small voltages, it is susceptible to electrical noise and result in incorrect readings from Analog Inputs. Excess cable length from users equipment can be a source of electrical noise. Shortening these runs will yield some improvement.

Input Modules should be located as close to the user equipment as possible. Extension ribbon cables to meet a system's distance requirement may be purchased from **TENOR CNTROLS**. Use part number 100-6-9080-XX, with XX defining the cable length. Maximum distance between Processor and last Module on the daisy-chain must not exceed 300 m (1000').

8. When memory protection or power back-up is needed, a 24VDC Back-Up Power Supply for the Processor and Remote Power Supply Modules may be ordered.

9. TENOR CONTROLS suggests fuses be installed in the following situations.

a. Input Modules should be externally fused on each common Input Line to protect Inputs from damage.

b. Output Modules should be externally fused on each common Output line to protect each Output from damage.

c. A Master Fuse should be installed to provide panel wiring protection for Inputs or Outputs sharing a common power rail.

10. Allow plenty of space for cable routing and trunking. Be sure that wiring to the Modules has sufficient slack, should the modules need rewiring. Include spare cables in wiring runs for future expansion and maintenance purposes.

3.2 ASSEMBLING THE SYSTEM

Before assembly, sure that you have read and understand the Safety Warnings stated at the beginning of this manual.

Processor mounting requires the use of the mounting brackets. It occupies the equivalent to four spaces on the mounting brackets. Brackets should be attached to a pair of 'DIN', 'C' type rails or other suitable chassis via the mounting holes in on brackets.

3.2 ASSEMBLING THE SYSTEM CONTINUED

Shunting brackets for Processor and Modules may be installed next to each other for compact installation.

Processor and Modules may be installed in any order desired. However, Module positioning is critical in electrically noisy environments. Care in placing them should be taken.

Mount the Processor and Modules by hooking the metal bar on rear of unit over top of mounting bracket and pivot unit down until its captive screws can be fastened. Turn the screws one quarter turn to secure.

3.3 TESTING OF THE PROCESSOR

The following bench level test procedure should be used to test and configure the entire system. This will allow system familiarization prior to full installation.

1. Disconnect Processor from all Modules. Install an End Link and connect power cable. Select 110 or 220 Volts AC on front panel voltage selector.

2. When power is present, the **Power On** indicator should light. The Watch Dog LED should be lit shortly afterward. The RUN light will also begin to flash on and off. The display should indicate ICC 100. If any of above fails to occur, refer to the Troubleshooting Techniques for assistance.

3. Connected the Processor to the Video Display Terminal or Hand Held Programmer (HHP) with a 100-6-9091 Editing Cable.

4. The Processor will require initialization of RS232 parameters. It should also be noted that the Video Terminal may require set-up. Refer to the appropriate manual. Use the following RS232 set-up format:

START BIT - 1 DATA BITS - 7
PARITY BIT - 1 STOP BIT - 1
BAUD RATE - BETWEEN 300 AND 4800 BAUD

The Processor is set at the factory to 1200 Baud and Odd Parity. However, the Processor Baud Rate and Parity can be reset by using the Engineers Panel.

5. Next, turn on the Video Terminal. The Processor should be on to initiate system. If the Processor is already ON, you must re-initiate the system by turning it OFF and ON, again.

3.3 TESTING THE PROCESSOR CONTINUED

The unit should display:

***READY**

If this does not occur, check cable connections. The end labeled DATA I/O must be connected to Processor. Check the RS-232 Pin Connections against the table in the Appendix 2 for correct wiring.

3.3 TESTING THE PROCESSOR CONTINUED

If failure to operate at this point still occurs, check for the proper set-up of the communication format and adjust as necessary.

If the above procedure does not solve the problem, please refer to the factory for guidelines.

6. When the ***READY** message appears, press ESC (Escape) on Video Terminal keyboard to reset system. Again, the display will show ***READY** on Video Terminal. The Processor is now operating in the **EDIT** mode as shown by the extinguished **RUN LED** on the front panel.

The User program may now be entered into the Processor. **Refer to Chapter 7 for programming details.**

3.4 MODULE INTERCONNECTION

Modules are interconnected via ribbon cables to carry power and data throughout the system. The Ribbon cable connectors on each Module have removable tabs. Note that the connectors are identical and polarized. Separating the tabs allows ejection of the plug.

Ribbon cable from the Processor must always be plugged into left connector of a Module. Similarly, the ribbon cable connected to right connector of any module must always be connected to left connector of next module. Starting with Processor, ribbon cables are plugged into successive sockets until last Module has been connected.

The End Link must always be installed in final Module of chain.

The Processor will sense a cable or circuit failure within 10 msec. If a failure occurs, the Watchdog shutdown procedure will be executed.

3.5 CABLING - MODULES TO USER EQUIPMENT

Each Module has detachable I/O terminal blocks. These blocks are UL reconized, heavy duty terminal connectors that enable quick connection and disconnection of external wiring.

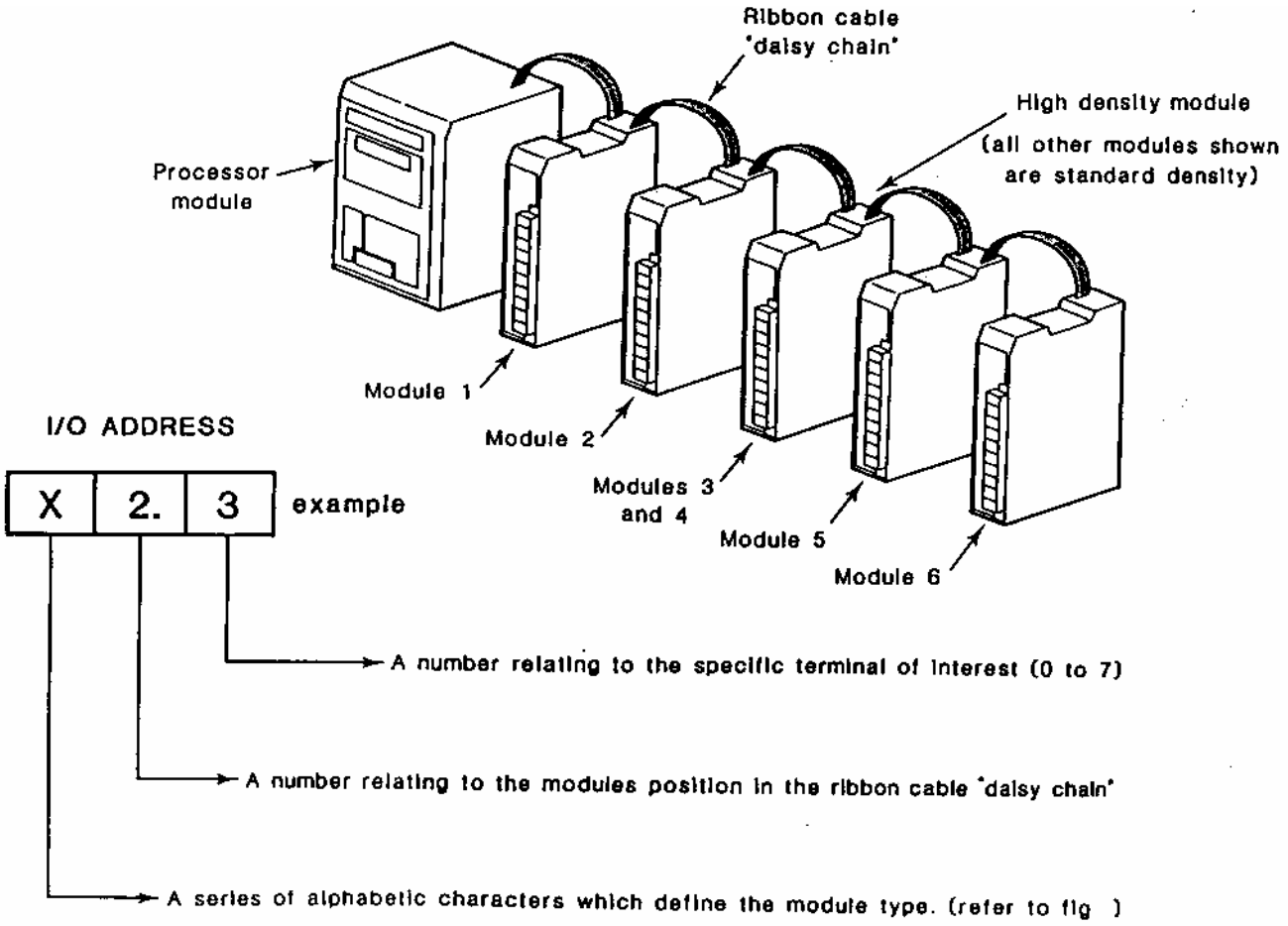
3.5 CABLING - MODULES TO USER EQUIPMENT CONTINUED

Modules may be removed without disturbing any cabling. To remove, unscrew the screws located immediately above and below the terminal block, and remove it.

Terminal blocks feature screw type wire terminals capable of holding up to two **1.0 mm2** wires. Approximately **8mm** of shielding should be stripped back from each cable before the wire is inserted under the washer for connection.

NOTE: Output Modules are not internally fused. Depending on the application's Output switching devices, it may be necessary to insert fuses on them.

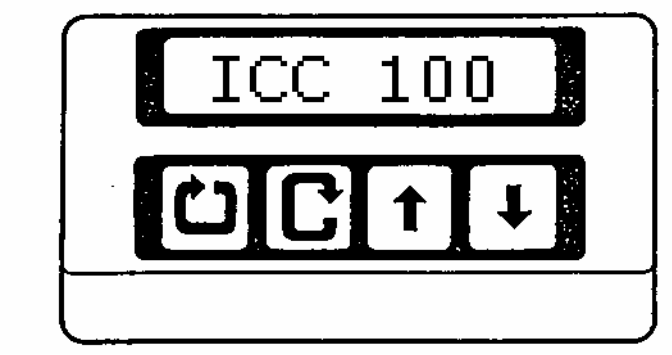
(Illustration of daisy chain)



4.0 ENGINEER'S PANEL

4.1 DESCRIPTION

The Engineer's Panel is located on the face of the Processor and includes an eight character LCD and four push button switches. Control of the Engineer's Panel is resident within the Processor in menu form to allow user to easily move to selected display option. The four buttons with legends are arranged as shown below:



The Engineer's Panel is a maintenance and debugging aid to examine variables, and I/O. It is also used for manually forcing I/Os and setting the RS232 baud parameters.

The Display is used to show the type of fault detected by the Processor. Pressing the four push buttons simultaneously will reset Processor.

The Engineer's Panel can be used to display output messages and push buttons used as inputs. It can be used to determine status of program steps, adjust timer/counter parameters and to perform general diagnostic tasks.

A new main menu option will be displayed each time button 1 is pressed. Holding the button down will automatically cycle through the options.

4.2 MENU ITEMS

DISPLAY	**INPUTS**	**FLAGS**	**VARIABLE**	**UTILITY**
TIM/CNT	**OUTPUTS**	**FIND**	**TIME**	

Each of these options with their respective sub-headings will be discussed in the order shown above.

4.2.1 DISPLAY

	Button 1	**Button 2**	**Button 3**	**Button 4**
LED Display	DISPLAY	ICC 100	VERSION	X.X

This option can be used at any time to display user generated status messages. The message will be displayed in sections if the it is greater than eight characters in length. Use Buttons 3 and 4 to move forward or backward through the message one word at a time.

4.2.2 TIMERS/COUNTERS

	Button 1	Button 2	Button 3	Button 4
LED Display	TIM/CNT	T/C 08	Increment Timer Counter	Decrement 0 to 63
		S08 5535	Set Point	0 to 9999
		U08 0000	Value Up	0 to 9999
		D08 0000	Value Down	0 to 9999
		08 U0 D0	Up/Down Signals	
		08 E0 R0	Enable/Reset Signals	
		08 O0 Z1	Time Output/Instantaneous Output	

Note: Only the SP value can be modified by Engineers Panel buttons, assuming the FORCE mode is enabled.

4.2.3 INPUTS

	Button 1	Button 2	Button 3	Button 4
LED Display	INPUTS	X000 OFF	Increment X0.0 to X31.7 & ON or OFF	Decrement

Status and value of Inputs can be monitored. The Processor RUN LED shows the ON/OFF state of an Input.

4.2.4 OUTPUTS

	Button 1	Button 2	Button 3	Button 4
LED Display	OUTPUTS	Y00.0 OFF	Increment Y0.0 to Y31.7 & ON or OFF	Decrement

Status and value of Outputs can be monitored and the Processor. The RUN LED shows the ON/OFF state of that Output.

4.2.5 FLAGS

	Button 1	Button 2	Button 3	Button 4
LED Display	FLAGS	F000 OFF	Increment Loc. 0 to 255 & ON or OFF	Decrement

Status and value of Flags can be monitored. The Processor RUN LED shows the state of a Flag. Flags cannot be Forced On or Off.

4.2.6 FIND

	Button 1	**Button 2**	**Button 3**	**Button 4**
LED Display			**Increment**	**Decrement**
	FIND	INPUTS		
		X00.5 ON	Input No.	X00.0 / X31.7
		10 or NO FIND	Adjust line # up or down.	
		First line	Forward or backward along the line a word at a time.	
	X00.5 ON	next line #40 or NO FIND	Adjust line # up or down.	
	FIND	INPUTS	OUTPUTS	
		Y00.7OFF	Output No.	Y00.00 to Y31.7
		30 or NO FIND	Adjust line # up or down.	
		First line	Forward or backward along the line a word at a time.	
	Y00.7OFF	next line #80 or NO FIND	Adjust line # up or down.	
	FIND	INPUTS	FLAGS	
		F045	Flag No.	F000 to F255
		60 or NO FIND	Adjust line # up or down.	
		First line	Forward or backward along the line a word at a time.	
	F0045	next line #230 or NO FIND	Adjust line # up or down.	
	FIND	INPUTS	TIM/CNT	
		T/C 05	Time/Counter #	00 to 63
		20 or NO FIND	Adjust line # up or down.	
		First line	Forward or backward along the line a word at a time.	
	T/C 05	next line #280 or NO FIND	Adjust line # up or down.	
	FIND	INPUTS	LINE	
		L06000	Adjust line # up or down.	
		First line	Forward or backward along the line a word at a time.	
		or NO FIND		

If **FORCE** is on, but not LOCK ON, (3) or (4) may be used to force Inputs or Outputs. See the description of **UTILITY** for details.

To search for an earlier parameter, in body of program, press Button 1 to cycle to end of the program. Holding the button down will cause the display to wrap around to the beginning.

4.2.7 VARIABLE

	Button 1	Button 2	Button 3	Button 4
LED			**Increment**	**Decrement**
Display	VARIABLE	A	Var. Letter	A to Z to A
		A (00)	Var. Number	00 to 85 to 00
		0000	Var. Value	0000 to 9999

The Processor can store up to 1365 or 4095 variables, depending upon memory configuration, and numeric variable locations. String variables identified with the symbol $ will not be displayed on the Engineer's Panel.

4.2.8 TIME

	Button 1	Button 2	Button 3	Button 4
LED			**Increment**	**Decrement**
Display	TIME			
		FRI 10.22	No function	
		05-08-87	No Function	
		MIN 22	Minute No.	00 to 59
		HOUR 10	Hour No.	01 to 23
		DATE 04	Day No.	01 to 31
		Month 05	Month No.	01 to 12
		DAY FRI	Day No.	MON to SUN
		YEAR 96	Year No.	
			0 to 9	00 to 99

The Real Time Calender and Clock provides full display of TIME, DATE, DAY, MONTH or YEAR. The values displayed can be adjusted if **FORCE** is enabled.

4.2.9 UTILITY

	Button 1		Button 2	Button 3	Button 4
				Increment	**Decrement**
LED	UTILITY	RS232			
Display			BAUD1200	Baud Rate	300 to 4800
			WIDTH	Width No.	10 to 100 characters.
			PARITY	EVEN / ODD / NONE	
	UTILITY		RS232	FORCE	
			FORCE ON	FORCE OFF / FORCE ON	
			LOCK	NO LOCK / LOCK	
			CLEAR	CLEAR / CLEAR	
	UTILITY		RS232	MISC.	
			99 P/C 4	% of memory available	
			08V 32P/16V 24P	Memory configuration	
			VERS 4.3	Version of Program	
			ECHO	NO ECHO / ECHO	

Any I/O can be **FORCED ON** or **FORCED OFF** without regard to the program status. Once **FORCE ON** has been enabled, the user can return to any of the Input, Output or Time main menu headings by pressing the left most button.

5.0 ACCESSORIES

5.1 100-3-9050 HAND HELD PROGRAMMER & MEMORY STORAGE DEVICE (HHP)

The HHP is a hand sized keyboard with an eight character alpha-numeric display, EPROM socket, cable and plug. It is used to program the **Series 100** Processors and store a program in an EPROM. The HHP plugs into the Processor data port for connection into the **Series 100** system.

This unit is capable of programming most 2764 type U-V erasable EPROM chips requiring a 21V programming pulse. EPROMS may be erased by exposing chip's window to ultra-violet light for the appropriate time.

5.1.1 OPERATION

The HHP is intended to function as if it were a VDU. It has the capacity to store and manipulate a single program line of up to 74 characters long. The display will only show 8 characters at a time.

Most of the 20 keys have more than one function. The normal function of any key is boldly printed in the center of its pad. The other functions are selectable by use of the UPPER or LOWER keys. To use the UPPER or LOWER shift functions, press the appropriate shift key followed by the specific UPPER or LOWER function desired. The shift mode keys are now reset to its normal mode. To enter additional shift commands, the User must use this two key combination each time an UPPER or LOWER command is needed.

NOTE: Whenever the UPPER or LOWER keys are used in conjunction with any function key, it will be presented as **(upper)RUN** or **(lower)LIST**, for example, in this manual.

The > and < keys are used to move display along line, right and left. Use **(upper)>** or **(upper)<** to move the display to one end or other of a line. The **E** key displays next line and **(upper) E** displays previous line.

5.1.1 OPERATIONS CONTINUED

The Processor must be in its EDIT mode before any communication with Processor can take place. To communicate, press **(upper) ESC**. The ***READY** message should be displayed. If this does not occur, see The **Series 100** Manual, Chapter 8, Troubleshooting Techniques. To Execute the program, **(upper) RUN** key should be pressed. One or more letters may be inserted anywhere in a line by positioning window. A flashing cursor indicates where next insertion will start if there are less than 8 characters in line. Insertion will commence at right end of display if there are 8 or more characters in line. Similarly, words may be deleted anywhere in line using **RUB** key. Press **E** key to send completed line to Processor. If no errors were encountered, the start of a line will be displayed, otherwise the right end of error message will be displayed.

All Boolean, BCD and Timer variables may be programmed using the HHP.

Note: Spaces are automatically inserted when programming.

Each line is limited to 72 characters, 8 commands. If capacity is exceeded error symbol **L** is displayed.

The **E** key must be pressed at end of every key sequence to tell the Processor the sequence is complete.

5.1.2 LOADING A PROGRAM INTO THE PROCESSOR

The HHP is used to load programs from an EPROM into the Processor. Following is the procedure to be used:

-Insert the programmed EPROM into the socket in the HHP.

-Press **(upper) LOAD** and **LOADING** will be displayed until programming is complete. When it has completed the task, **LOADED** will be displayed. Should the user program occupy more than one EPROM the display will indicate **NXT PROM** when the first EPROM has been loaded. Fit the next EPROM into the HHP and press **E** to continue loading.

-The following messages will indicate an unsuccessful loading of program.

FAILURE - indicates an invalid program in EPROM.

ERROR - indicates a communications error. The loading should be repeated.

ABORTED - unable to load program usually caused by lack of Processor RAM space.

5.1.3 SAVING A PROGRAM RESIDING IN THE PROCESSOR

The HHP can be used to save a program residing in the Processor. Following is the procedure to be used:

-The EPROM programmer will program most types of 2764 EPROMS requiring a 21V programming pulse (INTEL algorithm) providing an indefinite program storage capability.

5.1.3 SAVING A PROGRAM RESIDING IN THE PROCESSOR CONTINUED

-Insert the 2764 EPROM into the socket of the HHP.

-The display will show either **EMPTY** or **FULL** when **(upper) EMPTY** is pressed. **FULL** indicates that the EPROM has not been completely erased. Once **EMPTY** is displayed, continue as shown below.

-Press **(lower) SAVE** to initiate the programming procedure and the display will show **SAVING** until fully programmed. **SAVED** will be displayed when the task is complete. Should the program exceed the space available with one EPROM, the display will indicate **NXT PROM**. This indicates that the next empty EPROM should be inserted. Pressing the **E** key will continue the programming of the next EPROM.

-Two fault displays which can occur during the EPROM programming are:

FAILURE - indicates a faulty EPROM which should be replaced.

ERROR - indicates a communications error between the HHP and the Processor. Check the connections between these units, re-check the program and repeat the save operation.

-A previously programmed EPROM may be verified using SAVE; if it is identical, **SAVED** will be displayed.

5.1.4 SPECIFICATIONS

POWER SUPPLY - 5.7 to 10.5 volts at 100mA maximum.

INTERFACE - RS232, RS422 at 1200 baud, Odd parity bit, TXD and RXD

EPROM TYPE - 2764 EPROM (21V Vpp)

WEIGHT - 350 grams, (12.3 oz.)

SIZE - 145mm x 90mm x 32mm, (5.7"x 3.5"x 1.3")

ENVIRONMENTAL - 0 to 40°C.; 5-85% Relative Humidity, non-condensing.

FUNCTIONS - AND, OR, TO, NOT, X, Y, F, (,), TE, NR, CU, CD, ZO, OP, VU, VD, SP, WI, WO, BI, BO, 0 to 9, ., A, I, ESC, RUN, NEW, CLEAR, LOAD, SAVE, EMPTY, FIND, LIST, GOTO, +, -, *, / ,=

5.2 100-3-0946 EPROM MEMORY CHIP

Use 2764 or equivalent EPROM chips in Hand Held Programmer and Memory Storage Device.

5.3 100-3-9060 DIGITAL I/O SIMULATOR

The Digital I/O Simulator is used to simulate inputs and observe outputs in order to test application software. It behaves exactly as four 8 way Modules in series, each Module having both input switches and output LED indicators. The Simulator is connected to the rest of the controller system by the ribbon cable and may be used as a temporary alternative to any four adjacent 8 way (or two 16 way) I/O Modules.
Note: Inputs and Outputs in each row have the same address number.

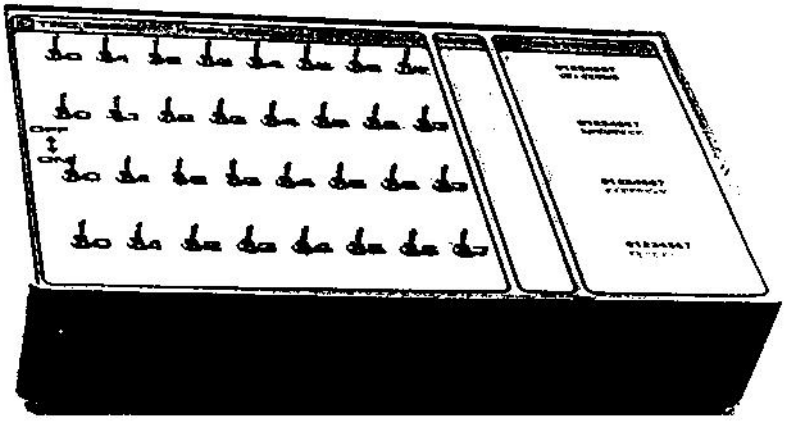
5.3.1 SPECIFICATIONS

SUPPLY LOAD - 38a units.

CASE MATERIAL - high impact plastic Front Panel - hard matte plastic

SIZE - 215mm x 45mm x 80mm, (8.5"x 1.7"x 3.1")

WEIGHT - 550 grams, (1.2 #)



5.4 100-3-9062 & 9063 OPERATOR CONTROL PANEL

5.4.1 DESCRIPTION

In larger programmable controller systems, there is often the requirement for the operator to be able to adjust several system parameters in addition to monitor system operation. In systems where the 9030 Thumbwheel Module or the 9031 LED Display Module are not appropriate, the 9062/9063 Operator Control Panel can be used for greater capability.

The 9062 Operator Control Panel comprises a numeric keypad, 6 user defined pushbuttons, 6 user defined indicators, (four pushbuttons and indicators on the 9063) and a 20 character alpha-numeric display. In addition, two 5-24VDC inputs with optional switch contact energize and two 0.5A 240VAC relay outputs are provided. The 16 conductor ribbon cable interface is used to connect the panel to the rest of the **Series 100** System.

The panel may either be portable or mounted in a panel cutout. The case has gasketed seals to prevent contamination.

Operation of the panel allows an operator to select one of six categories, enter a category indentification number to display the required parameter or message. This parameter could then be modified by entering a new value via the keyboard. System messages may be displayed at any time.

To the programmer, the model 9062 appears as two modules, 16 bits wide.

5.4.1 DESCRIPTION CONTINUED

The lower module handles display and keyboard data. Display data may be a numeric, string constant or variable. For example, using the ODISPLAY 1 "HELLO", ODISPLAY 1 A(B,C,D), or ODISPLAY 1 $A command. Numeric data may be input to a variable only using the OINPUT 1 A(B,C,D) or OINPUT 1 $A commands. Numeric data is entered on the keypad and terminated with the ENTER key. If an entry error is made prior to pressing the ENTER key, the CLEAR key will cancel for re-entry. When making a numeric entry, the display will show the number being entered.

The upper module is used as an input and output module sharing the same address. The inputs are taken from the user defined buttons and the two utility input connections. The outputs go to the indicators and two relay outputs under user control.

Below is detailed an example of a part of a typical software driver for the panel:

```
100 TASK 6 EVERY 500
110 ODISPLAY 1 "READY"
120 IF NOT X11.0 THEN GOTO 240 !LOOK FOR FUNCTION KEY
121 ! ENTER VALUE KEY PRESSED
122 ON Y11.0 !LAMP ON
130 ODISPLAY 10 "ENTER SENSOR NUMBER (1 TO 9)";
140 OINPUT 10 A  !GET NUMERIC INPUT
150 IF A LT 1 THEN GOTO 130 !CHECK LOWER LIMIT
160 IF A GT 9 THEN GOTO 130 !CHECK UPPER LIMIT
170 ODISPLAY 10 "SENSOR";  !"SENSOR 1 = 1234"
171 ODISPLAY 10 A;
172 ODISPLAY 10 "=";
173 ODISPLAY 10 B(A)
180 IF X11.1 OR X11.2 OR X11.3 OR X11.4 OR X11.5 THEN GOTO 120
190 ODISPLAY 10 "ENTER NEW VALUE";
200 OINPUT A(2)
210 IF A GT 999 THEN GOTO 190  !LIMIT UPPER VALUE
220 IF A LT 0 THEN GOTO 190 !LIMIT LOWER VALUE
230 B(A) = A(2)  !LOAD NEW VALUE
231 OFF Y11.0  !LAMP OFF
     ...
```

5.4.2 SPECIFICATIONS

Display: 20 characacters, 9mm height, 4 LEDs user definable.

Keyboard: 0 to 9, CLEAR, ENTER, 6 user definable keys; model 9063 has 4 user definable keys.

Inputs: two each, 4.5 to 24 V AC/DC. Threshold 2.5V typ. at 2.0mA.
"commoned" with user defined switches.

Outputs: two NO relay contacts, rated 120VAC 0.5A; 24VDC 1.0A resistive load, with LED indication. (LED indication is not included on the 9063).

Isolation: 1500VAC for 1 minute.

5.4.3 COMMANDS

The following describes the operation of the OINPUT and ODISPLAY commands when used with the 9062/9063 operator interface module.

The 9062/9063 occupies two module address locations, the upper being associated with switch inputs and relay outputs. The lower module address is used to transfer data to and from the display and keypad inputs of the module. Because two way communications exist between the **Series 100** and the **9062/9063** it is important that the force function is not used to change any of the bit locations on either the input or output of the module address or that other commands are used to change the module address.

The position of the ODISPLAY and OINPUT commands within the program is not important but because of the execution times they should be placed in lower priority tasks. If a CLEAR statement is included at the start of a program it is necessary to include two task breaks before an OINPUT or ODISPLAY command is executed.

Here is a sample program.

```
10 CLEAR
20 GOTO 30
30 GOTO 40
40 ODISPLAY 1 "Ready"
```

To enable OINPUT and ODISPLAY commands to be places in different tasks, a number of control bytes are transmitted to indicate that access to a particular module address is in progress.

These are as follows;

00 - Module address free for transfer.
01 - Enable keypad input on 9062/9063.
02 - Transmitted by ODISPLAY to get the module address.

5.4.3.1 ODISPLAY

For ODISPLAY, one character at a time is set on the module address and a break initiated. When character has been echoed from the 9062, the next character is sent to the module address. The minimum time required for the transmission of a string is the number of characters times 20ms.

If the command is terminated with a semi-colon, then the last byte to be sent will be 00. Otherwise, a carriage return followed by a 00 is transmitted. The carriage reutrn instructs the 9062 to display any characters previously sent. Because a break occurs after each character task, stack space is required to save parameters. The depth of the stack required for each case of ODISPLAY is given below.

5.4.3.2 ODISPLAY X "String Constant"

```
10 ODISPLAY 1 "String constant"
```

This command outputs the string constant directly to the module address/ Task Stack Depth: 3 Bytes.

5.4.3.3 ODISPLAY X STRING VARIABLE

```
10 ODISPLAY 1 A$
```

The string variable is sent directly to the module address. Because the command takes a number of passes through the task to execute, care should be taked that the string variable being accessed is not changed by other parts of the program during this period. The string variable should also avoid containing any of the control bytes used by the interface.

Task stack depth: 6 bytes

5.4.3.4 ODISPLAY X NUMERIC VARIABLE

```
10 ODISPLAY 1 A
```

This command shares a common buffer with the PRINT statements. The system will wait until any PRINTS in progress have been completed before the data is transferred to the module address. During the command, other PRINT statements are inhibited.

Task stack depth: 9 bytes

5.4.3.5 OINPUT

This command first transmits a prompt to the 9062 and then enables keypad input by transmitting to 01 control character. The system will wait until the enter key is pressed on the 9062. At this time, if other input statements are in progress, the OINPUT command will wait until the INPUT buffer becomes available. Otherwise, the incoming data is loaded into the INPUT buffer. The disabling of other INPUT statements occurs and the information is saved to the variable location. On completion of a transfer, the INPUT buffer is released and the module control byte is set back to 00. The input will occur over several passes of the task. The task stack depth required for various commands is given below.

5.4.3.6 OINPUT X STRING VARIABLE

```
10 OINPUT 1 A$
```

Task stack depth: 3 bytes.

5.4.3.7 OINPUT X NUMERIC VARIABLE

```
10 OINPUT 1 A
```

Task stack depth: 7 bytes.

5.5 100-3-9070 AUXILLARY OR REMOTE POWER SUPPLY

The Auxillary Power Supply may be used to provide power to I/O Modules in larger systems or where Modules are placed in a remote location from the Processor. Line losses would prevent supplying power from the Processor. The power supply may be powered from 110/240VAC or 24VDC. It provides a NO pair of contacts for Watchdog use via the screw terminal connector. The same supply loading factors apply to this unit as with the 100-6-9001-XX Processor.

5.5.1 SPECIFICATIONS

AC POWER SUPPLY - 110, 220 or 240VAC +10%, -15%, 47 - 63Hz at 150 VA max via IEC connector. A ground to internal chassis is provided near IEC connector.

LOW VOLTAGE - Two screw terminals may be used as a 24VDC over-voltage supply.

DC SUPPLY - may be used with a protected and fused supply input or as a current limited 24VDC output when using an AC power supply.

Supply Input - 24VDC +15%, -20% at 3.0A max.

Supply Output - 24VDC ± 20% current limited to 200mA, typical.

SUPPLY LOAD - Capable of driving multiple I/O Modules with a combined load of no more than 300 'a' units or 200 'b' units.

5.5.2 ENVIRONMENT

Operating temperature - 0 to 60 ° C.

Storage temperature - -40 to 95 ° C. Humidity - 10% to 95% continuous, non-condensing.

Vibration - 10 to 55Hz, 1.5mm dual amplitude.

Shock - 10G, 11msec

5.5.3 WATCHDOG OUTPUT

One NO contact with overvoltage protection which closes when interface to Processor is functioning correctly.

Contact rating - 2A at 240VAC/30VDC

Output voltage - 264VAC max.

Isolation Voltage - 1500VAC, 1 min.

5.5.4 SIZE

195mm x 230mm x 190mm max.
7.7"x 9.0"x 7.5"

5.5.5 WEIGHT

3.7 Kg,
8.17 #

5.5.6 CONNECTIONS

1 - +24V normal supply I/O.
2 - OV supply.
3 - Watchdog NO contact. 4 - Watchdog NO contact.

Contact factory for information.

5.6.1 100-6-9086 MODULE END LINK

Supplied with Processor.

5.6.2 100-3-9087 POWER SUPPLY CORD

Included, less plug on one end.

5.6.3 100-3-9088 MOUNTING BRACKET

Supplied with Processor and Modules.

5.6.4 100-6-9089 MOUNTING RAILS

2 each 480 mm, (18.9") with 20 Mounting Bracket Adapters, included.

5.6.5 SERIAL PRINTER - ORDER A STANDARD PC PRINTER

5.6.6 VIDEO DISPLAY TERMINAL AND KEYBOARD

5.7 CABLES

100-3-9080-02 CABLE - 2" Processor to Module or Module to Module.

Length - 51 mm (2"), Supplied with Processor and Modules

STANDARD RS-232 CABLE

Use to connect Processor to printer or Video Display.

100-6-9080-XX CABLE ASSEMBLY

Non-standard length, for Processor to I/O Module or Module to Module. Lengths specified in 1 ft. increments.

100-6-9080-12 CABLE ASSEMBLY

Length - 12"

100-6-9080-24 CABLE ASSEMBLY

Length - 24"

100-6-9080-60 CABLE ASSEMBLY

Length - 60" (5')

100-6-9080-120 CABLE ASSEMBLY

Length - 120" (10')

6.0 INPUT AND OUTPUT MODULES SPECIFICATION SHEETS

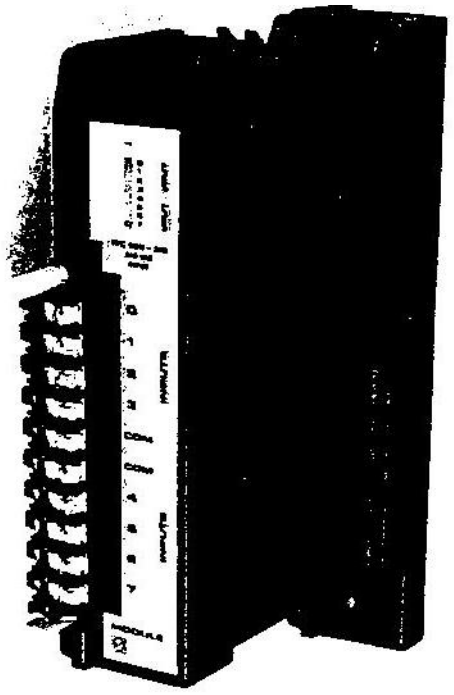
6.1 100-7-9011-XX INPUT MODULES

These Modules have eight general purpose inputs for use with all types of switches and on/off sensors. Each of the eight inputs are organized as two groups of 4 inputs. Each input has full wave rectification, opto-isolation, contact bounce protection, noise suppression circuitry and a green LED input status indicator.

SECIFICATIONS	**9011-01**	**9011-02**	**9011-03**	**9011-04**
RATED VOLTAGE	24 VAC/DC	115VAC	230VAC	12VAC/DC
OVERVOLTAGE	69 VAC/DC	200VAC	345VAC	30VAC/DC
MAX. 1 SEC.				
ON VOLTAGE	17-30 VAC/DC	80-160VAC	160-280VAC	9-14VAC/DC
OFF VOLTAGE	6VAC/DC	40VAC	80VAC	4VAC/DC
INPUT CURRENT	10mA typ. @ 24VAC/DC, 30mA max.	10mA typ. @ 115 VAC, 30mA max.	10mA typ. @ 230VAC, 30mA max.	10mA typ. @ 12VAC/DC 30mA max.
TURN ON TIME	8.5msec typ. 15 msec max. @ 24VAC/DC. at 60 Hz	8.5msec typ. 15 msec max. @ 115VAC at 60 Hz	8.5msec typ. 15 msec max. @ 230VAC at 60 Hz	8.5msec 15msec max @ 2VAC/DC at 60 Hz
TURN OFF TIME	12msec typ. 16msec max. @ 24VAC/DC. at 60 Hz	12msec typ. 16msec max. @ 115VAC at 60 Hz	12msec typ. 16msec max. @ 230VAC at 60 Hz	12msec typ 16msec max @ 12VAC/DC at 60 Hz
FREQUENCY	0 - 100 Hz.	47-63Hz	47-63Hz	0-100 Hz.
ISOLATION (1 min)	1.5kVAC	1.5kVAC	1.5kVAC	1.5kVAC
SUPPLY LOAD	15a units	15a units	15a units	15a units

CONNECTIONS

1-Input 0	3-Input 2	5-Common	7-Input 4	9-Input 6
2-Input 1	4-Input 3	6-Common	8-Input 5	10-Input 7



6.2 100-7-9016-XX INPUT MODULES

These Modules appear as two adjacent 8 point Input Modules to the Processor providing a higher density of 16 Inputs with one common for all 16 Inputs as shown in the CONNECTIONS section. Each Input point is full wave rectified, noise suppressed, optically isolated and has an LED status indicator.

SPECIFICATIONS	**9016-01**	**9016-02**	**9016-04**
RATED VOLTAGE	24 VAC/DC	115VAC	12VAC/DC
OVERVOLTAGE (MAX. 1 SEC.)	69 VAC/DC	200VAC	30VAC/DC
ON VOLTAGE	17-30 VAC/DC	80-160VAC	9-14VAC/DC
OFF VOLTAGE	6VAC/DC	30VAC	4VAC/DC
INPUT CURRENT	8mA typ. at 24VAC/DC, 20mA max.	8mA typ. at 115 VAC, 20mA max.	8mA typ. at 12VAC/DC 30mA max.
TURN ON TIME	10 msec typ. 20 msec max. at 24VAC/DC. at 60 Hz	10 msec typ. 20 msec max. at 115VAC at 60 Hz	10 msec typ. 20 msec max. at 12VAC/DC at 60 Hz
TURN OFF TIME	15 msec typ. 25 msec max. at 24VAC/DC. at 50 Hz	15 msec typ. 25 msec max. at 115VAC at 50 Hz	15 msec typ. 25 msec max. at 12VAC/DC at 50 Hz
FREQUENCY	0 - 63 Hz.	47-63Hz	0 - 63 Hz.
ISOLATION (1 min)	500kVAC	500VAC	500VAC
SUPPLY LOAD	25a units	25a units	25a units

Note: Contact TENOR CONTROLS for information and availability of a High-Speed version of these three Modules.

CONNECTIONS:

1 - Input 0		9 - Input 0	
2 - Input 1		10 - Input 1	
3 - Input 2		11 - Input 2	
4 - Input 3	<- First	12 - Input 3	<- Second
5 - Input 4	Address	13 - Input 4	Address
6 - Input 5		14 - Input 5	
7 - Input 6		15 - Input 6	
8 - Input 7		16 - Input 7	

17 Common for all Inputs

Note: The First Address is the Module address of upper 8 Inputs, the Second Address is the Module address of lower 8 Inputs. The second is always First+1.

6.3 100-7-9021-03 RELAY OUTPUT MODULE

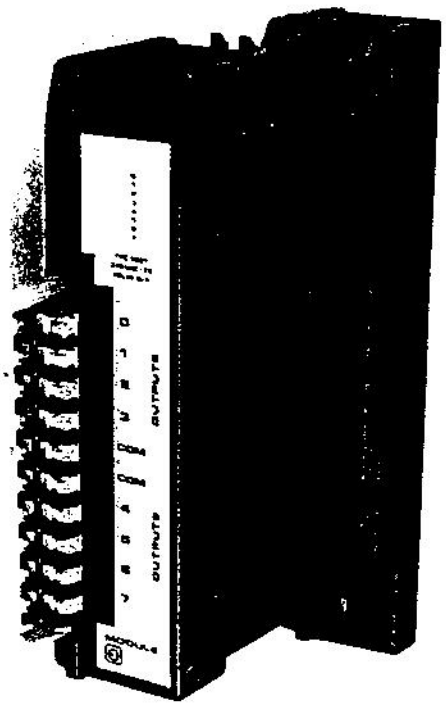
This Module provides eight normally open relay contacts in two groups of four, each group having a common return. Each contact has Varistor transient protection for low electrical noise emission, long contact life with inductive loads and a red LED status indicator.

SPECIFICATIONS

OUTPUT VOLTAGE - 240VAC maximum at rated current.
264VAC (353VDC) maximum.
CONTACT RATING - 2A at 240VAC or 30VDC (resistive load) with all 8 outputs driven. Each output may be used up to 10A and 240VAC/30VDC provided that the total for either group of four outputs does not exceed 10A.
MECHANICAL LIFE - 20,000,000 operations min., per output at no load.
CONTACT LIFE - 200,000 operations min., per output at 10A at 240VAC.
100,000 operations min., per output at 10A at 30VDC, resistive load.
DELAY TIME - 10msec typical at 20 ° C (24VDC supply).
ISOLATION VOLTAGE - 1.5kVAC for 1 minute.
SUPPLY LOAD - 10a units, 15b units.

CONNECTIONS

1 - Output 0		7 - Output 4
2 - Output 1	5 - Common for outputs 0 to 3	8 - Output 5
3 - Output 2	6 - Common for Outputs 4 to 7	9 - Output 6
4 - Output 3		10 - Output 7



6.4 100-7-9022-XX DC OUTPUT MODULES

These Modules feature semiconductor output switches providing faster and clearer switching characteristics. It has eight outputs arranged in two groups of four where, each group has a common return terminal. Each output has a normally open semiconductor contact driven from individual opto-isolators and has a LED status indicator. Fuse protection is best provided outside the Module for ease of access. An external power supply must be used to power the isolated output switch circuitry.

This Module is designed for use with DC switching applications where,

9022-01 - the **N variant** has a common negatve terminal.

9022-02 - the **P variant** has a common positive terminal.

Transient over voltage protection is incorporated to provide reliable operation with inductive loads. Be sure to make connections with the proper polarity.

SPECIFICATIONS

VOLTAGE RANGE -	+50VDC max
OUTPUT CURRENT AT 40 ° C -	0.010A min 2.0A max
OUTPUT CURRENT AT 60 ° C -	1.5A max
SURGE CURRENT (1:10 DUTY FACTOR) -	15A/1sec
ON VOLTAGE AT 2A/20 ° C -	0.65V typ 0.85V max
OFF LEAKAGE AT 20 ° C -	1.0mA max at 50V
TURN ON TIME AT 60Hz -	< 1msec
TURN OFF TIME AT 60Hz -	< 1msec
ISOLATION (1 MIN) -	1.5kVAC
SUPPLY LOAD -	15a

CONNECTIONS

1-Output 0	3-Output 2	5-Common	7-Output 4	9-Output 6
2-Output 1	4-Output 3	6-Common	8-Output 5	10-Output 7

6.5 100-7-9023-XX TRIAC OUTPUT MODULE

These Modules feature semiconductor output switches having eight outputs arranged in two groups of four where, each group has a common return terminal. Each output has a normally open semiconductor contact driven from individual opto-isolators. All outputs feature individual LED status indicators. Fuse protection is best performed outside the Module for ease of access.

These Modules are designed for use with AC switching. Output switching is undertaken at the zero voltage transition of the AC supply to insure lowest electrical noise. Snubber networks and MOV transient suppressors are provided to insure reliable operation with highly inductive loads. The snubber networks, in order to function correctly, pass a leakage current of up to 2.0mA at 240VAC and care should be taken to insure that this does not present a safety hazard.

SPECIFICATIONS

VOLTAGE RANGE - 100-3-9023-02	24 to 160 VAC
100-3-9023-03	160 to 240 VAC
OUTPUT CURRENT AT 40 ° C -	0.05A min 2.0A max
OUTPUT CURRENT AT 60 ° C -	1.2A max
SURGE CURRENT (1:10 DUTY FACTOR) -	15A/1sec
ON VOLTAGE AT 2A/20 ° C -	1.2VAC typ 2.0VAC max
OFF LEAKAGE AT 20 ° C -	6.0mA max at 260VAC
TURN ON TIMER AT 60HZ -	10msec max
TURN OFF TIME AT 60Hz -	10msec max
ISOLATION (1 MIN) -	1.5kVAC
SUPPLY LOAD -	20a

CONNECTIONS

1-Output 0	3-Output 2	5- -ve	7-Output 4	9-Output 6
2-Output 1	4-Output 3	6- -ve	8-Output 5	10-Output 7

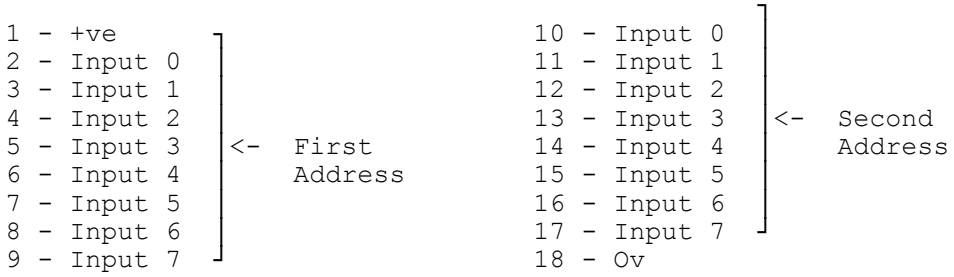
6.6 100-7-9024 SEMICONDUCTOR OUTPUT MODULE

This Module provides high density semiconductor switching outputs. These Modules appears as two adjacent 8 point Output Modules to the Processor providing a higher density of 16 low current DC Outputs with one common negative return for all 16 Outputs as shown in the CONNECTIONS section. All outputs feature over voltage and reverse voltage protection to insure reliable operation with inductive loads. An external power supply must be used to power the isolated output switch circuitry.

SPECIFICATIONS

OUTPUT (AND SUPPLY) VOLTAGE -	7.5V to 40VDC.
OUTPUT CURRENT -	120mA max at 60 ° C.
SURGE CURRENT -	400mA for 1 second 1:10 duty factor.
ON VOLTAGE DROP -	1.3V max (1.0V typ) at 25 ° C.
OFF LEAKAGE CURRENT -	100,µA max at 40V and 60 ° C.
TURN ON/TURN OFF TIME -	Less than 1msec.
ISOLATION -	500VAC for 1 minute.
SUPPLY LOAD -	15a units.

CONNECTIONS



Note: The First Address is the Module address of upper 8 Inputs, the Second Address is the Module address of lower 8 Inputs. Second is always First+1.

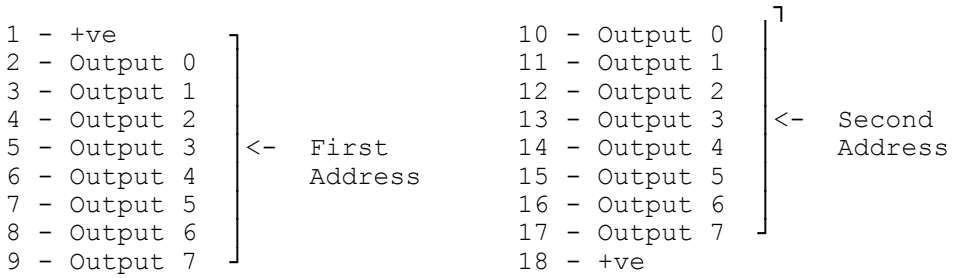
6.7 100-7-9025 HI CURRENT SEMICONDUCTOR OUTPUT MODULE

This Module provides high density semiconductor switching outputs. These Modules appears as two adjacent 8 point Output Modules to the Processor providing a higher density of 16 open collector Outputs that are arranged in two groups of eight Outputs. The switching voltage and current ranges offered by this Module allow a wide variety of loads to be driven. Each output features transient voltage suppression and a LED status indicator. An external power supply must be used to power the isolated output switch circuitry.

SPECIFICATIONS

OUTPUT VOLTAGE -	50VDC max, -0.5VDC min.
OUTPUT CURRENT -	1.0A DC max at 40 ° C.
	0.75 ADC max at 60 ° C.
SURGE CURRENT -	4.0A DC for 1 sec, 1:1 duty cycle
ON VOLTAGE -	0.7V typ, 0.85V max at 1.0ADC
	at 20 ° C.
OFF LEAKAGE -	1.0mA max at 50V at 20 ° C.
DELAY TIME -	1 msec typical.
ISOLATION VOLTAGE -	500VAC for 1 minute.
SUPPLY LOAD -	20a units.

CONNECTIONS



Note: The First Address is the Module address of upper 8 Inputs, the Second Address is the Module address of lower 8 Inputs. Second is always First+1.

6.8 100-7-9027, 9028 240 VAC RELAY OUTPUT MODULE

These Modules are an 8 point Output Module providing a higher density of eight contacts arranged as two groups of four, each group having commoned wiper contacts. Each contact has Varistor transient protection for low electrical noise emission, long contact life with inductive loads and a red LED status indicator.

SPECIFICATIONS

		9027	**9028**
OUTPUT TYPE	-	FORM C	FORM A

OUTPUT VOLTAGE - 240VAC maximum at rated current.
264VAC (353VDC) maximum.
CONTACT RATING - 2A at 240VAC or 30VDC, resistive load, 8 outputs on
Each output may be used up to 10A and 240VAC/30VDC
provided that the total for either group of four
outputs does not exceed 10A.
MECHANICAL LIFE - 20,000,000 operations min., per output, no load.
CONTACT LIFE - 200,000 operations min. per output at 10A and 240VAC.
100,000 operations min. per output at 10A and 30VDC
resistive load.
DELAY TIME - 10msec typical at 20 ° C (24VDC supply).
ISOLATION VOLTAGE - 1.5kVAC for 1 minute.
SUPPLY LOAD - 10a units, 15b units.

CONNECTIONS

1 - Common for y.0 to y.3			10 - Output y.4 NO 4	Second Address
2 - Output y.0 NO 0	First Address		11 - Output y.4 NC 4	
3 - Output y.0 NC 0			12 - Output y.5 NO 5	
4 - Output y.1 NO 1			13 - Output y.5 NC 5	
5 - Output y.1 NC 1			14 - Output y.6 NO 6	
6 - Output y.2 NO 2			15 - Output y.6 NC 6	
7 - Output y.2 NC 2			16 - Output y.7 NO 7	
8 - Output y.3 NO 3			17 - Output y.7 NC 7	
9 - Output y.3 NC 3			18 - Common for y.4 to y.7	

Note: The First Address is the Module address of upper 8 Outputs, the Second Address is the Module address of lower 8 Outputs. Second is always First + 1.

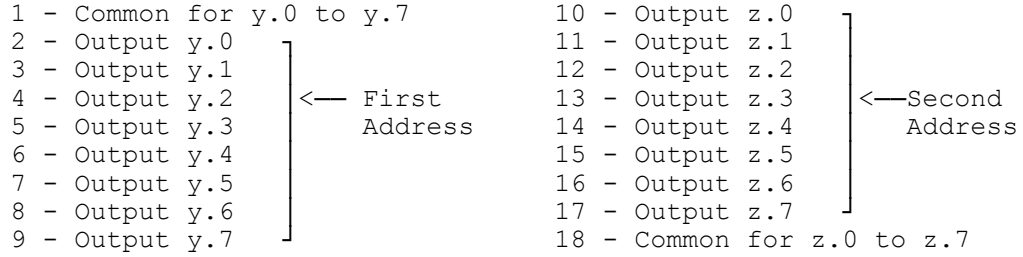
6.9 100-7-9029 AC RELAY OUTPUT MODULE

This Module appear as two adjacent 8 point Output Modules to the Processor providing a higher density of 16 relay Outputs. The output current rating of 1 Amp at 240VAC/30VDC allows a variety of loads to be driven. This Module does not have output transient voltage suppression, but includes a LED status indicator.

SPECIFICATIONS 9029

OUTPUT VOLTAGE -	240VAC max at rated current. 280VAC (390VDC) max
CONTACT RATING -	1A at 240VAC or 30VDC max. resistive load
MECHANICAL LIFE -	10 million operations min.
EXPECTANCY	per output
ELECTRICAL LIFE -	100,000 opereration min. per output
EXPECTANCY	at 240VAC,resistive load
DELAY TIME -	15msec typical, 30msec max.
ISOLATION VOLTAGE -	1.0kVAC for 1 minute.
SUPPLY LOAD -	10a units, 30b units.

CONNECTIONS



Note: y is Module address of upper 8 outputs, z is Module address of lower 8 outputs. z is always y+1.

6.10 100-7-9030 THUMBWHEEL SWITCH MODULE

This Module allows numeric data entry into the system. It comprises a PCB assembly with four 33mm high thumbwheels switches. IDC sockets at either end of the Module allow it to be linked to any other Module via the 16 way ribbon cable interface. This allows a considerable reduction in the cabling usually associated with these devices. This Module appears to the programmer as two adjacent 8 point input Modules to which are connected four BCD digits.

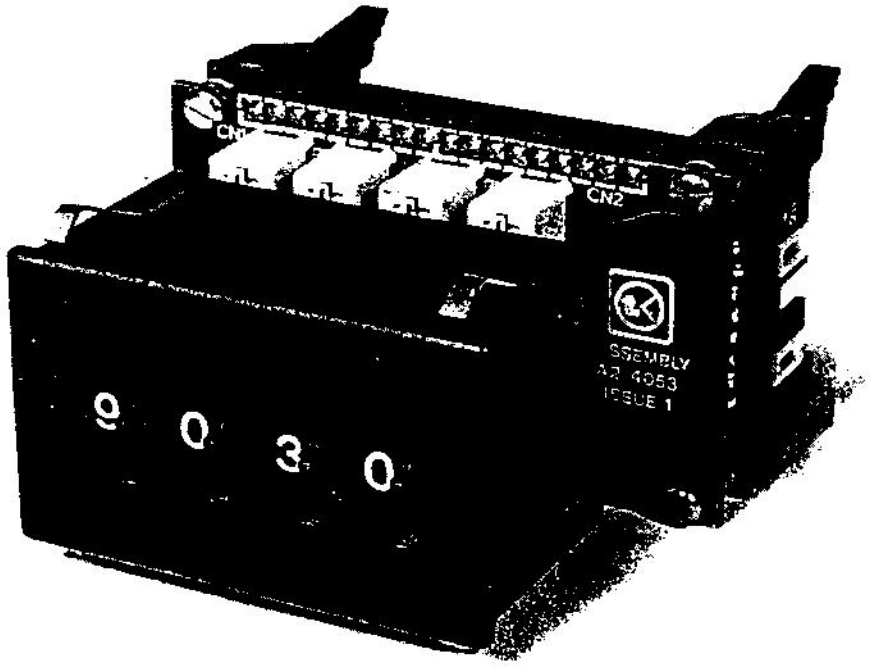
This Module requires a cable, see Accessory Section.

SPECIFICATIONS

4 Digit thumbwheel switch Module.
Supply load of 8 a units.
Panel cut-out 51 x 31mm.
Each BCD digit occupies 4 single digital input points.

Programming Example: 100 SP12 = W15.0

This load setpoint 12 with four BCD digits located at modules 5 and 6.



6.11 100-7-9031 FOUR DIGIT LED DISPLAY MODULE

This Module allows numeric data display on a front panel. It comprises a PCB assembly with four sockets to take standard 33mm high seven
segment display packages. Two IDC sockets allow the Module to be linked to any other via a 16 way ribbon cable interface. This allows a considerable reduction in the cabling usually associated with these devices.

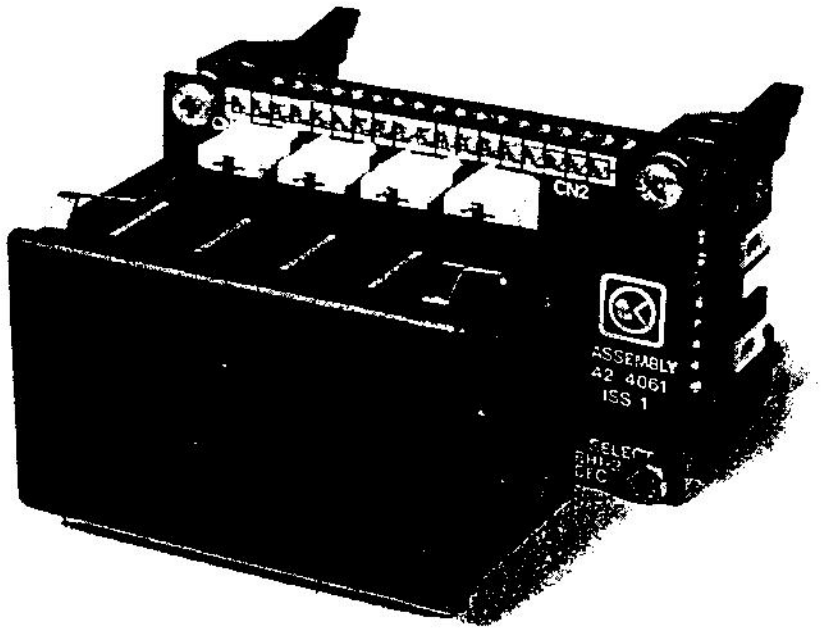
This Module appears to the programmer appears as two adjacent output Module locations which drive four BCD digit display packages.

SPECIFICATIONS

4 Digit LED display Module.
Supply load of 25a units.
Panel cut-out 51 x 31mm.
Each BCD digit occupies 4 single digital Output addresses.

Programming Example: 110 WO7.0 = VU12

Display will show Value Up for Timer/Counter 12 as 4 BCD digits at Module location 7 and 8.



6.12 100-7-9043-01, 02, 03, 9041-08 ANALOG INPUT MODULES

These Modules are an analog interface that may be simply connected to a wide range of sensors to measure parameters such as temperature, velocity, or flow. Low drift input amplifier circuitry allows direct interface with negligible drift effects. A current limited, 12V/100mA power supply output is isolated from the rest of the system (but with common 0v inputs) is provided to optionally power sensors or current loops. They are designed to replace P/N's 100-7-9041-XX.

SECIFICATIONS	9043-01	9043-02	9043-03	9041-08
CHANNELS -	4	4	4	1
SCAN TIME -	40 msec	40 msec	40 msec	10 msec
OUTPUT -	0-10VDC	0-1.0VDC	0-20mA	0-10VDC
SENSITIVITY -	±10V	±1.0V	±20mA	±10V

Common to all models:

UPDATE TIME PER CHANNEL-	80msec
INPUT IMPEDANCE -	1 Megohm
SERIES MODE REJECTION -	40dB at 50(60)Hz ±1%
COMMON MODE REJECTION -	86dB at 50(60)Hz ±1%
RESOLUTION -	12 Bit
ZERO DRIFT -	0 ± 1 unit at 25 ° C
TEMPERATURE DRIFT -	± 30ppm/ 1 degree C
ISOLATION -	±12.0V max. between channels
500VAC	1 min. between inputs and system
ENERGIZATION -	12.0V ±5% at 100mA max via Ve contact
SUPPLY LOAD -	20a units.

CONNECTIONS

1 - Ve +ve energization output channel 0	10 - Not used
2 - IN+ non-inverting input channel 0	11 - Ve +ve energization output channel 2
3 - IN- inverting input channel 0	12 - IN+ non-inverting input chan. 2
4 - GND Analog command ground	13 - IN- inverting input channel 2
5 - Ve +ve energization output channel 1	14 - GND Analog common ground
6 - IN+ non-inverting input channel 1	15 - Ve +ve energization output channel 3
7 - IN- inverting input channel 1	16 - IN+ non-inverting input chan. 3
8 - GND Analog common ground	17 - IN- inverting input channel 3
9 - Not used	18 - GND Analog common ground.

```
Programming Example: 120 A = AIN4.0       !Channel 0 to variable A
                     130 A(2)  = AIN 4.1  !Channel 1 to variable A(2)
```

Numbers read from a channel range from ± 4096. However, full-scale deflection is calibrated to ± 4000.

6.13 100-7-9043-09 TRUE RMS ANALOG INPUT MODULE

This Module generates the true RMS value corresponding to the input voltages which may vary over a wide frequency range. It is intended for use in the measurement of AC voltage, current, or power. The four inputs all have the same sensitivity, and their ground line is commoned and each input has a grounded 10 ohm shunt which may be connected across the input for current measurement.

SPECIFICATIONS

UPDATE TIME PER CHAN.- 10msec
INPUT SENSITIVITY - 2.000V, or 100mA using internal shunt for output of 4000. Other sensitivities between 100mV and 10V built to order.
INPUT IMPEDANCE - 500k ohms.
OUTPUT - 0-1.0 VDC
ACCURACY - ±1.0% of reading (at 25 ° C)
FREQUENCY RESPONSE - DC to 10kHz for specified accuracy.
CREST FACTOR - 1 to 4 for specified accuracy; 7 for -1% of reading.
SETTLING TIME - 1.5sec to 99% of step value.

CONNECTIONS

1 - Not connected	10 - Not connected
2 - IN+ input channel 0	11 - Not connected
3 - IN- 10 ohm to analog ground	12 - IN+ input channel 2
4 - GND Analog common ground	13 - IN- 10 ohm to analog GND
5 - Not connected	14 - GND Analog common GND
6 - IN+ input channel 1	15 - Not connected
7 - IN- 10 ohm to analog ground	16 - IN+ input channel 3
8 - GND Analog common ground	17 - IN- 10 ohm to analog GND
9 - Not connected	18 - GND Analog common GND

6.14 100-7-9043-10, 11, 12 THERMOCOUPLE INPUT MODULES

These Modules are designed to directly interface with different types of thermocouples. Cold junction compensation is provided via an ambient temperature sensor embedded in the connector. The input will be biased out of range in the event of a cable break. Each input also has an internal link which may be set for UPSCALE or DOWNSCALE break detection.

All inputs in the Module are isolated from the rest of the system. They operatewithin a restricted voltage range (±12VDC or peak AC) from each other to allow the use of non-isolated thermocouples on a common (grounded) metallic vessel. One of the Module GND terminals should be connected to the vessel being sensed in order to reduce any risk of errors due to electromagnetic interference.

SPECIFICATIONS:

SENSOR -	9043-10	J type Thermocouple
	9043-11	K type Thermocouple
	9043-12	T type Thermocouple
UPDATE TIME/CHANNEL -	10 msec	
INPUT IMPEDANCE -	1 Megohm across inputs, 100 Megohm to system GND	
LEAD RESISTANCE -	100 ohms maximum.	
FULL SCALE SENSITIVITY -	50mV	
CALIBRATION -	0 output at 0° C	

Output	Voltage	Temperature 9043-10		9043-11		9043-12	
		°C	°F	°C	°F	°C	°F
1000	12.5	231	448	307	585	259	498
2000	26.0	457	854	602	1115	467	873
3000	37.5	674	1245	904	1660	-	-
4000	50.0	870	1598	1232	2250	-	-
Suggested Scale Factor:		.225	.4235	.304	.576	.246	.467

CONNECTIONS

1 — Ve +ve energization output channel 0
2 - IN+ non-inverting input channel 0
3 - IN- inverting input channel 0
4 - GND Analog command ground
5 - Ve +ve energization output channel 1
6 - IN+ non-inverting input channel 1
7 - IN- inverting input channel 1
8 - GND Analog common ground
9 - N/C (cold junction sensor)
10 - N/C (cold junction sensor)
11 - Ve +ve energization output ch.2
12 - IN+ non-inverting input ch.2
13 - IN- inverting input channel 2
14 - GND Analog common gnd.
15 - Ve +ve energization output ch.3
16 - IN+ non-inverting input ch.3
17 - IN- inverting input channel 3
18 - GND Analog common gnd.

6.15 100-7-9041-20, 21, 22, and 24 RTD SENSOR INPUT MODULES

These Modules provide four inputs for 3 wire RTD sensors with line compensation. Inputs are isolated from the rest of the system, but not from each other. As the sensor is energized, any lead break will cause an upscale error, and lead short will usually cause a downscale error.

SPECIFICATIONS

SENSOR TYPE - Platinum, 100 ohm 3 wire to DIN 43760 or BS1904.
UPDATE TIME/CHANNEL - 10 msec
EXCITATION CURRENT - 0.5mA nominal.
EXTERNAL LOAD - 100 ohms maximum in 3 wire mode.
RESISTANCE
TEMPERATURE RANGES - 100-7-9041-20 0 to +700°C
100-7-9041-21 -90 to +100°C
100-7-9041-22 -180 to +200°C
100-7-9041-24 -200 to +400°C
ACCURACY AT 20 DEG C - ±0.3%, ±1 count.

OUTPUT	TEMP. °C			
100-7-9041-	20	21	22	23
- 4095			DOWNSCALE SHORT	
- 4000		-97	-187	(-200=-2200)
- 1000		-25	-49	-92
- 500		-12.5	-25	-47
- 100		- 2.5	- 5	-10
0	0	0	0	0
100	17.5	2.5	5	10
200	35	5	10	19
500	87.5	12.5	24	48
1000	175	25	49	97
2000	350	50	98	195
3000	525	75	149	295
4000	700	100	200	400
4095			UPSCALE BREAK	

Suggested Scale Factors:

100-7-9041-21	0.0246
100-7-9041-22	0.0484
100-7-9041-22	0.0960

CONNECTIONS

1 - Ve +ve energization (B) channel 0
2 - IN+ sense input (A) channel 0
3 - IN- -ve energization (C) channel 0
4 - GND Analog common ground
5 - Ve +ve energization (B) channel 1
6 - IN+ sense input (A) channel 1
7 - IN- -ve energization (C) channel 1
8 - GND Analog common ground
9 - Not connected
10 - Not connected
11 - Ve +ve energization (B) channel 2
12 - IN+ sense input (A) channel 2
13 - IN- -ve energization (C) channel 2
14 - GND Analog common ground
15 - Ve +ve energization (B) channel 3
16 - IN+ sense input (A) channel 3
17 - IN- -ve energization (C) channel 3
18 - GND Analog common ground

6.16 100-7-9042-11, 12, 14, 21, 22 & 24 ANALOG OUTPUT MODULES

These Modules provide analog outputs with 12 bit resolution suitable for interfacing to motor drives. These outputs are isolated from the system (but not each other) and do not require user power. A current limited 12V/100mA supply output isolated from the rest of the system (but with common 0v to analog outputs) is available for driving external devices. Modules are beingreplaced by P/N's 100-3-9044-x4.

SPECIFICATIONS

# OF OUTPUTS -	9042-11 - One Analog Output
	9042-12 - Two Analog Outputs
	9042-14 - Four Analog Outputs
	9042-21 - One Analog Output
	9042-22 - Two Analog Outputs
	9042-24 - Four Analog Outputs
OUTPUT -	9042-11,12 & 14 - 0 to 10V at 20mA maximum, 2.5mV/bit
	9042-21, 22 & 24 - 0 to 20mA current sink, 5 A/bit
MAXIMUM LOAD - RESISTANCE	450 ohm with internal loop supply 900 ohm with external 24V loop supply.
COMPLIANCE -	V loop-6V
ACCURACY -	± 0.3%
SCAN TIME -	40msec (or four scans)
ISOLATION -	500VAC for 1 minute
SUPPLY LOAD -	20a units
UPDATE TIME/ -	10 msec / CHANNEL

CONNECTIONS

1 - Ve +ve power output channel 0	10 - N/C
2 - OUT-output channel 0	11 - Ve +ve power output channel 2
3 - GND Analog common ground	12 - OUT-output channel 2
4 - N/C	13 - GND Analog common ground
5 - N/C	14 - N/C
6 - Ve +ve power output channel 1	15 - N/C
7 - OUT-output channel 1	16 - Ve +ve power output channel 3
8 - GND Analog common ground	17 - OUT-output channel 3

Programming Example: 120 AOT5 = A(1), B(2), B(3), Z(5,4,3)

A number output to each channel may be within the ± 4096 range. Full-scale calibration occurs for outputs of ± 4000 and will require a scalingfactor to yield real units. A scale factor of 200 will correct a 20 mA value to yield a full-scale output of 4000 in 100-3-9042-2x modules.

6.17 764-7-0013 4 Channel Temperature Module

Description: This temperature module has four individually adjustable channels controlled by one temperature probe. Each channel has a "Form C" relay output internally fused at 5 Amps. The probe temperature is displayed by an external volt meter. The scaling of millivolts can be directly read as ° Farenheit. The module may be used as a stand alone unit or connected to the **Tenor Series 100 Industrial Control Computer.**

Operation: Each channel can be set individually by adjusting the multi-turn potentiometers through a hole on the front panel. The temperature range can be adjusted by turning the pot fully CCW to sense 32° F and fully CW to sense 212° F. When the temperature of the probe reaches the set temperature of a channel, the LED, Relay and Input for that channel turn on. When the temperature has fallen below the set point, the LED, Relay, and Input for the channel turns off. The hysteresis potentiometers allow a 2 to 8 degree range. Channels 1 to 4 can be set via P7 to P10, respectively. These pots are set to to approximately 4° Farenheit of Hysteresis at the Factory. The settings may be altered form inside the temperatur module.

Connections

Top Connector	**Bottom Connector**
Probe (+)	#1 - NC Relay Contact Meter
Meter (+) = 1	#1 Common Meter
Meter (+) = 2	#1 NO Relay Contact
Probe (-)	#2 NC Relay Contact
L1 120 VAC = Terminal 1	#2 Common
L2 120 VAC = Terminal 2	#2 NO Relay Contact
	#3 NC Relay Conact
	#3 Common
	#3 NO Relay Contact
	#4 NC Relay Contact
	#4 Common
	#4 NO Relay Conact

6.18 764-7-0014 Temperature Probe

Description: The temperature probe allows the measurement of temperature in gas or liquid. The probe measures current variations and interprets them into an absolute temperture change. This unit can operate DC supply voltage between +4 and +30 Volts. Its accuracy is ±1.8° F.

Operation: This temperature probe has a high impedance output and insensitive to voltage drops over lines. This probe may be used in a variety of applications. It performance range is recommended for -67 to + 302° F.

6.19 100-3-1005 Temperature Readout

Description: This three digit, LED readout is designed to work with the 100-7-0014 Temperature Probe, displaying temperature in °F.

Operation: The readout requires a +5 VDC supply and connection to a temperature probe for operation.

Connections:

1. N.C.	6. (+) Sensor
2. ° Offset Scaling	7. Power GND
3. Signal GND	8. + 5 VDC
4. ° Gain Scaling	9. Offset Calibration
5. (-) Sensor	10. N.C.

6.20 100-7-8028 3 Channel Liquid Level Module

Description: The 8028 module has 3 individual liquid level sensor inputs. Each sensor input has a separate high and low level probe connection and a latching "Form C" Relay output internally fused to 5 Amps. This module may be used as a stand alone unit or with the **Tenor Series 100 Industrial Control Computer.** The liquid level inputs are addressed as Xm.0, Xm1 and Xm.2.

Operation: The sensor sends an 8 KHz, 8 VAC signal. This signal is passed through the Ground probe and the High or Low probe. The module determines the presence or absence of liquid by comparing the resistance between probes to the reference of an internal resistance circuit. Each channel of the module has an adjustable sensitivity setting which allows the channel to detect the presence of liquids with upto 50 K Ohms between the probes. Sensitivity can changed by adjusting a pot through a window on the front panel. Turning fully CCW can detect the liquid with the highest resistance.

Single Probe Operation: When a single probe application is required use only the hogh probe and ground. Connect the ground to the side of a metallic tank or the bottom of a non-metallic tank. When the liquid reaches the high probe, the relay and input turn on for that specific channel and remain on until the level drops below the high probe.

Dual Probe Operation: Two probes are needed if an application calls for a relay and input to be turned on when the liquid reaches the high probe and to turn off the relay and input when liquid drops below the lower probe.

Note: When used with the **Series 100 Processor,** power must not be applied to the Processor before applying power to the liquid level probe.

Connections:

1.) #1 NC Relay Contact
2.) #1 Common
3.) #1 NO Relay Contact
4.) #1 Low Input
5.) #1 High Input
6.) #2 NC Relay Input
7.) #2 Common
8.) #2 NO Relay Contact
9.) #2 Low Input
10.) #2 High Input
11.) NC Relay Contacts
12.) #3 Common
13.) NO Relay Contact
14.) #3 Low Input
15.) #3 High Input
16.) Ground (Common)
17.) L2 120 VAC
18.) L1 120 VAC

6.21 100-7-8030 4 Channel Temperature Module with 0 to 4 Slot Eliminator

Description: The 8030 Temperture Module has four individually adjustable channels, each controlled by one temperature probe. Individual channels have a "Form C" relay output internally fused at 5 Amps. The probe temperature is displayed by an external volt meter. Millivolts can be directly read as ° Farenheit. This module may be used as a stand alone unit or connected to a **Tenor Series 100 Industrial Control Computer**. The module takes up one 8 way I/O slot. The digital status of temperature inputs are addressed as Xm.0, Xm.1, Xm.2 and Xm.3. The I/O slot eliminator circiut can be reset to take up one to four slots. The module is factory set to have zero positions eliminated.

Operation: Each channel can be set individually by adjusting the multi-turn pots through the window on the front panel. The temperature range can be adjusted by turning the pot fully CCW to sense 32° F and fully CW to sense 212° F. When the probe reaches the set temperature of a specific channel, the LED, Realy and Input for that channel turns on. When the temperature drops below the set point, the LED, Relay and Input turn off. The Hysteresis Potentiometers allow a 2 to 8 degree range. The point may be set through pots P5 to P8 for channels 1 to 4, respectively. A Hysteresis setting of approximately 4° F is made at the factory. It may be altered by adjusting the settings inside the module.

I/O Slot Eliminator: Each I/O slot eliminator simulates the presence of one 8 way module. It maintains the program and integrity of the Shur-Manage system when less than a full compliment of modules are used. Setting the number of I/O slots to be eliminated is done by setting the five position DIP switch SW1 inside the module. Only **ONE** switch may be **ON** at a time. Switch should be performed when power to the **Series 100** Processor is **OFF**.

SW1	SW2	SW3	SW4	SW5	# of I/O SLOTS ELIMINATED
ON	OFF	OFF	OFF	OFF	0
OFF	ON	OFF	OFF	OFF	1
OFF	OFF	ON	OFF	OFF	2
OFF	OFF	OFF	ON	OFF	3
OFF	OFF	OFF	OFF	ON	4

Connections:

1.) #1 NC Relay Contact
2.) #1 Common
3.) #1 NO Relay Conatct
4.) #2 NC Relay Conatct
5.) #2 Common
6.) #2 NO Relay Contact
7.) #3 NC Relay Contact
8.) #3 Common
9.) #3 NO Relay Conatct
10.) #4 NC Relay Contact
11). #4 Common
12.) #4 NO Relay Contact
13.) Probe (+)
14.) Meter (+)
15.) Meter (-)
16.) Probe (-)
17.) L1 120 VAC
18.) L2 120 VAC

6.22 100-7-9044 HIGH SPEED ANALOG OUTPUT MODULES

DESCRIPTION

The **SERIES 100** provides four channel outputs with 12 bit resolution on the 100-7-9044-XX modules. The outputs are isolated from the system but not from adjacent outputs.

A current limited 12V/100 mA supply output isolated from the system but connected to the Analog Output Common is available for driving external devices or powering current loops. On power up, all outputs are forced to 0 V.

Analog output data cannot be directly manipulated. It should be loaded from variables of the relevant module. For example:

```
120 AOT5 = A, B(2), B(3), Z(5,4,3)
130 AOT6 = A, A, 0, 2047
```

Unipolar modules accept values from 0 to 4095 and are calibrated at 4000. Bipolar modules accept values from - 2047 to + 2047 and are calibrated at 2000. Values exceeding the acceptable range will be set to 0 V.

Output Values:

9044-1X	0 to 10V at 10mA max 1k ohm min, 2.5mV/bit (0-4000)
9044-2X	4 to 20mA sink 450 ohm max, 4ua/bit (0-4000)
9044-3X	-4 to -20mA source 450 ohm max, 4ua/bit (0-4000)
9044-4X	-10 to +10 V at 10mA max 1k ohm min 5mV/bit (-2000 to +2000)

Compliance: Vloop - 2.5 V typical

Accuracy: 0.3%

Scan Time: 10 msec (or 1 scan)

Isolation: 1.5 KVAC for 1 minute

Supply Load: 20 a units

Connections:

1.	Ve +ve Channel 0	10.	N/C
2.	OUT Channel 0	11.	Ve +ve Channel 2
3.	GND Analog Common Ground	12.	OUT Channel 2
4.	N/C	13.	GND Analog Common Ground
5.	N/C	14.	N/C
6.	Ve +ve Channel 1	15.	N/C
7.	OUT Channel 1	16.	Ve +ve Channel 3
8.	GND Analog Common Ground	17.	OUT Channel 3
9.	N/C	18.	GND Analog Common Ground

7.0 SERIES 100 PROGRAMING LANGUAGE

7.1 DESCRIPTION

7.1.1 GENERAL

The **Series 100** Industrial Control Computer System is an I/O computer designed for control of industrial machine and process systems. It must be able to simultaneously control many independent system activities. It must rapidly respond to a change in status of inputs. These responses are dependent upon several factors.

1) The program response as dictated by the user.
2) The system response as directed by the control software.

The **Series 100** Processor uses a language that allows the program to be written with standard Boolean statements and compiled for high speed execution. They are written such that they may be repeatedly scanned to give the appearance of parallel execution. For sequential or numerical operations, sections may be written using control oriented BASIC statements. These statements are compiled into an intermediate form to allow fast execution without sacrificing memory space. Boolean statements act upon image register data. This data is updated with physical I/O status prior to a scan to provide unambiguous operation.

7.1.2 PROGRAMMING

The 100-3-9050 Hand Held Programmer (HHP) and the 100-3-8006 Video Display Terminal and Keyboard (Terminal) provide two methods of programming the **Series 100** Processor.

The HHP is restricted to Boolean programming. However, it has the capability to list and enter programs. The HHP includes a built-in EPROM programmer. The EPROM programmer allows programs to be more permanently stored in EPROMs.

The HHP and Terminal communicates with the Processor using the RS232 protocol. The terminal can accept Baud rates between 300 and 4800 bits per second. The HHP has a fixed rate of 1200 baud.

The Terminal provides full access to the Editor, 22 lines of immediate program display and full use of Basic Commands.

When a Terminal is connected to the Processor with a 100-3-9091 Editing Cable, the Terminal display should respond with:

***READY**

This is followed by a cursor symbol. This indicates where the next Keyboard entry will start. After entering each line the Carriage Return (CR) key should be pressed. This signals the Editor that the line is complete. If the line structure is correct, the Cursor will automatically move to the start of
the next line.

7.1.2 PROGRAMMING CONTINUED

If an error is detected by the Editor during line interpretation, an error message will be displayed. Programs are created by the use of command statements and values. Each program consists of a series of instruction lines made up of:

Program lines that commence with a unique line number in the range of 0 to 65,534.

Required expressions and statements with spaces inserted between elements of a statement.

Comments in upper or lower case characters may be added to a line by using an exclamation mark **!** before the comment. This informs the system that the following information is not part of the program.

Length of a line is restricted to less than 250 memory locations after processing by the Editor. This means that some type of program line content may not be as long as others. Line numbers increasing by 10 each time gives the user the option of adding extra lines as a result of program error or modification. The Processor works through the program in the order of line number, not in the order in which they were entered. Adding comments to the program as it is generated provides the program with a permanent record of what each line is intended to do. Memory space will be used to store these comments which could restrict available program space. A sensible compromise would add comments for understanding and in areas of complicated programming.

Valid lines of program are those which are and contain correct commands and statements. Generally, the lines of the program will appear to be logical if they have been correctly generated. The line Editor will check this. However, the Editor cannot check the way the lines of program interact when the program is running. Therefore, there are two basic rules which should be observed during program creation to avoid confusion.

1) A good understanding of the available commands will help in defining what is possible.

2) Flow charts or similar techniques will aid in program creation.

With these tools, the tasks required of a program are separated into manageable parts. Each part is considered separately.

7.1.3 I/O ADDRESSING

The Processor transfers or gathers data to and from Modules using a structured address or label. This address is used during programming to define the source or destinction of data required for the application. This address comprises three items of information namely:

a) The Module Type.
b) The Module Location.
c) The Terminal or Channel Number.

7.1.3.1 THE MODULE TYPE

The Module type is an alphanumeric label which defines its type, Input or Output Module. It also defines part of the I/O address:

```
X  - Single Digital Input
Y  - Single Digital Output
MI - Byte Digital Input
MO - Byte Digital Output
BI - Byte of BCD Input
BO - Byte of BCD Output
WI - Word of BCD Input
WO - Word of BCD Output
AIN - Analog Input
AOT - Analog Output
```

The WI and WO labels are used to read from Thumbwheel Switch Module or write to 4 digit Display Module.

7.1.3.2 THE MODULE LOCATION

The location is derived from Modules position in daisy-chain. The Processor is the first Module in that daisy-chain and has location '0'. The next Module will then be Module 1 and so on. Where high density Modules are used they occupy the equivalent of two normal Module positions and have two Module locations.

7.1.3.3 THE TERMINAL OR CHANNEL NUMBER

Channel number is determined by that terminal or channel number to which the connection is made. The numbers are located on the I/O Module front panel. The complete expression of an I/O address should be entered via the Terminal in this form:

```
I/O Module type, Module location and terminal number.
```

For example, X1.6 is a digital Input from Module number 1, Input terminal 6.

7.2.THE EDITOR

The **Series 100** has a powerful Editor which is used during program creation and Processor operations.

7.2.1 PRINCIPLE EDITOR COMMANDS

CLEAR - resets all Boolean and numeric variables to zero, reset values of Timer/Counters and clear all Flags.

ESCAPE - keyboard key which will halt a running program and enable the Editor only if Editor Link is in place.

LIST(CR) - lists the program.
LIST(Line Number)(CR) - lists a single line, specified by line number.
LIST(line number-)(CR) - lists the program starting with that line number.
LIST(line number-line number)(CR) - lists program range starting with the first numbered line and ending with the last numbered line.

NEW(CR) - erases all previous programs. Due to potential severity of this command Editor will query the command by displaying DELETE YES/NO. If the NEW function is chosen, the response must be Y (CR), if the response is N the old program will remain.

RUN - suspends the EDITOR, and cause the users program to be executed from the first line. To restore the EDITOR, an ESC should be entered. It is necessary to execute a RUN command after the last program change prior to removal of the programming device. Otherwise, GOTO and GOSUB addresses will not have been inserted, and on power up a N/A FAIL will occur.

STOP - pauses the program during RUN. I may be temporarily inserted into a program to aid debugging. When executed, if a programming device is connected, application program execution will cease, a "STOPPED AT LINE xxxx" message will be sent to the programming device, and the Editor will be entered. If STOP is executed without a programming device connected, it will be ignored.

7.2.2 EDITOR CONTROL CAPABILTIES

The Editor is designed to allow easy program generation and modification with a series of Editor control capabilities that are available.

CURSOR KEYS - move the Editor's cursor along a line and up and down lines allowing easy access to areas of program modification.

The following commands all require the Keyboard control key (CNTL) to be depressed at the same time as the stated character and are called control characters.

(CNTL)**X** - cancels a line only during line entry.

(CNTL)**A** - opens a line at cursor position allowing insertion of new program information. A second **(CNTL)A** will show whole line including inserted information.

7.2.2 EDITOR CONTROL CONTINUED

(CNTL)**R** - redisplays a line which was disallowed by Editor due to an error. Cursor position will now indicate position on the line of the error.

There are other Editor commands which are very useful when modifying parts of a program such as:

DELETE(line number)(CR) - deletes line number specified.

DELETE(line number-)(CR) deletes from specified line # to end of program. The Editor will question multi-line deletes with the **DELETE YES/NO** prompt.

DELETE(line number-line number)(CR) - deletes more than one line including the second line number. If the two line numbers are not consecutive then blocks of program lines may be deleted. The Editor will question multi-line delete with the **DELETE YES/NO** prompt.

EDIT(line number) displays line specified and invokes line editor.

The user may then move cursor along line to point where the change is required. Character immediately to left of cursor may be removed using RUB key, or character immediately over cursor may be changed by typing over it.

To make an insertion of extra characters, cursor should be placed at position where change is required; pressing **Control A** displays line up to cursor and additional characters may be inserted. A second **Control A** will display the whole of new line and a (CR) will then re-enter it. Also a (CR) may be entered to re-submit the line after insertion without viewing.

FIND - Searches for variable or statement specified. On CR, Terminal will indicate the next occurrence in the program of that variable or statement.

LOAD(CR) - Sends object code from EPROM in HHP to Editor from and loads it to non-volatile program memory. Command will abort if parity, checksum or invalid character error occurs.

RENUMBER(CR) - renumbers all program lines in increments of 10.

SAVE(CR) - downloads object code in Editor, via data port to EPROM, to HHP for storage. Once started, all of current program will be transmitted unless aborted by an ESC.

SP - list Setpoints Specified. Two number separated by a comma will list a range. CR will list Set Point(s) requested.

VU - Displays value of Value Up as specified Timer/Counter(s).

Note: The Editor will recognize a command/statement in abbreviated form. Two examples are:

REN-RENUMBER CL-CLEAR

7.3 BOOLEAN LOGIC PROGRAMMING

This section is separated into two main areas of interest, Logic Expressions and Boolean Programming. Logic Experessions deals with Boolean concepts and sections starting with Boolean Programming deals with the capabilities of the **Series 100** system for Boolean operation and the conversion of traditional control circuitry into Boolean terms. The use of Boolean operators may be freely mixed with Basic operators provided the line was correctly and constructed.

7.3.1 LOGIC EXPRESSIONS

Logic control circuits are based upon the use of solid state switches or gates. These gates can only have two states, open or closed. By combining several gates together in particular patterns it is possible to represent the 3 common logic expressions:

OR - Either of several Inputs will produce an Output.

AND - Only when all Inputs are present will an Output be produced.

NOT - The presence of an Input produces the lack of an Output.
Conversely, an Output will be present if there is no Input.

These logical expressions are best illustrated by using a symbolic notation. Boolean is such a notation to express logical concepts in an algebraic form. Boolean algebra is a symbolic method of examining logical relationships. All logical statements can only have two forms: TRUE or FALSE. This can be related directly to the state of a switch, **OFF** or **ON**; giving a signal of 0 or 1, respectively.

Boolean algebra is based on three symbols:

+ means OR A + B means A or B.

· means AND A · B means A and B.

¯ means NOT $\bar{A}$ + B means NOT A or B.

These symbols have no connection with their conventional mathematical meanings. Remember that the symbol = is used to separate Inputs from Outputs in a logical statement.

For example, A·B = Z where A and B are Inputs and Z is the Output.

The easiest way of understanding a Boolean expression is with the use of Truth Tables. These simply show how the Output of a Boolean expression relates to its Inputs. Truth tables are drawn on a grid arrangement and the following examples show such tables for the **OR, AND** and **NOT** functions. The Inputs to the Boolean expression use the letters **A** and **B** and the Output uses the letter **Z.**

A OR B = Z. This is the logical representation.

A + B = Z. This is the Boolean representation.

A	B	Z
0	0	0
0	1	1
1	0	1
1	1	1

The truth table shows that an Output exists whenever Input A or B or both are a logic 1.

A AND B = Z. This is the logical representation.

A · B = Z. This is the Boolean representation.

A	B	Z
0	0	0
0	1	0
1	0	0
1	1	1

The truth table shows that an Output exists whenever Input A and Input B are at a logic 1.

NOT A AND B = Z. This is the logical representation.

$\overline{A}$ · B = Z. This is the Boolean representation.

A	B	Z
0	0	0
0	1	1
1	0	0
1	1	0

The truth table shows that an Output exists when the "inverse of A" and Input B are a logic 1.

Longer expressions can be expressed in exactly the same way except more input terms must be considered. For example, the following truth table shows the Output Z for all permutations of Inputs A, B and C where:

A and B and C = Z This is the logical representation.

A · B · C = Z This is the Boolean representation.

A	B	C	Z
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

This truth table shows the Output state, Z, for the AND of the three Inputs A, B, and C.

Note: For more complex arrangements which use many separate AND, OR terms grouped together to describe a complete control function, the separate terms should be isolated between brackets.

A AND B OR C = Z

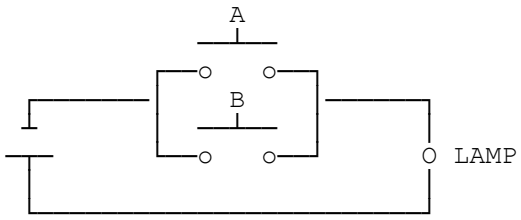
This does not state which parts should be considered together. If we were to logically read the above expression we would conclude that 'A AND B' should be evaluated before considering the 'OR C' term. This is exactly what the Processor would interpret from the above. However, we really meant that B or C should be considered separately. The term should have isolated those terms with brackets.

A AND (B OR C) = Z

This is quite a different expression to that above.

Each of the Boolean expressions can be represented using real elements such as switches and lamps. The three examples discussed previously using two Inputs A, B and an Output Z are shown below where the Inputs are switches and the Output is a lamp.

A + B = Z

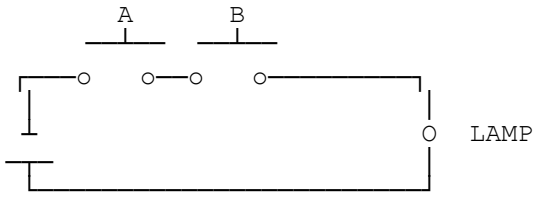


A	B	Z
0	0	0
0	1	1
1	0	1
1	1	1

The truth table shows that an Output exists whenever Input A or B or both are a logic 1.

In this case the switches are normally open and will be termed, at logic zero when open. Switch A or switch B will illuminate the lamp Z. Note from the truth table and from the circuit above that to turn off the lamp requires both switches to be opened.

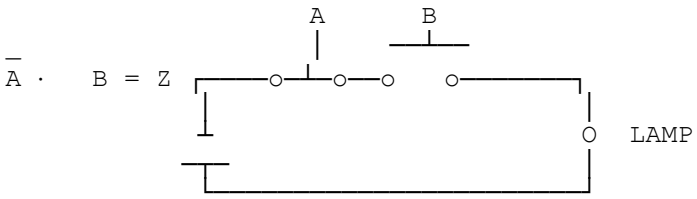
A · B = Z



7.3.1 LOGIC EXPRESSIONS CONTINUED

A	B	Z	The truth table shows
0	0	0	that an Output exists
0	1	0	whenever Input A and
1	0	0	Input B are at a
1	1	1	logic 1.

Here we see that both A and B must be closed for the lamp to illuminate. Note from the circuit above and, the previous truth table any single switch being opened will turn off the lamp.



A	B	Z	The truth table shows
0	0	0	that an Output exists
0	1	1	when the "inverse of
1	0	0	A" and Input B are a
1	1	0	logic 1.

This circuit shows the 'A' switch as being normally closed and is expressed as a NOT function. Here we see that with A being normally closed it would only be necessary to close B to light the lamp.

The use of alphabetical characters to define Inputs and Outputs is just for convenience and, to establish a common method of representation. When programming the above expressions we have to replace the Input and Output characters by real Inputs and Outputs. If switch A were monitored by Input X1.3, switch B were monitored by Input X1.4, and the Output were driven by Y2.1 then the function A · B = Z would have the program line:

```
10 NOT X1.3 AND X1.4 TO Y2.1
```

7.3.2 BOOLEAN PROGRAMMING

The **Series 100** Boolean software allows program creation using fundamental commands and statements:

AND - where I/O are evaluated in series.

OR - where I/O are evaluated in parallel.

NOT - the inverse function.

TO - used to assign an Input Expression to an Output expression.

All of these commands can be used to define a program line using Input and Output as terms to be manipulated.

7.3.2 BOOLEAN PROGRAMMING CONTINUED

The following example shows how the expression

$$A \cdot \overline{(B + C)} \cdot D = Z$$

The above expression can be converted into a program.

The following addresses will be given to the terms of the equation:

```
Input  A  -  X1.3    Input  D  -  X2.5
Input  B  -  X1.4    Output Z  -  Y5.2
Input  C  -  X5.7
```

The program line would be written:

```
10 X1.3 AND NOT (X1.4 OR X5.7) AND X2.5 TO Y5.2.
```

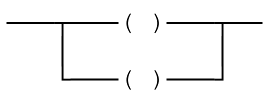
The words of the **Series 100's** Boolean language are:

AND, OR, NOT, TO, ON, OFF

The above words are equivalent to the following ladder diagram connections.

AND —] [—] [—

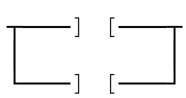
TO ——] [——( )—



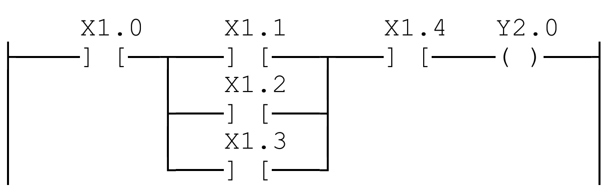
NOT —]/[— Input

NOT —(/)— Output

OR



In addition to the above language words, parentheses can be used to break down a network into sub-sections.

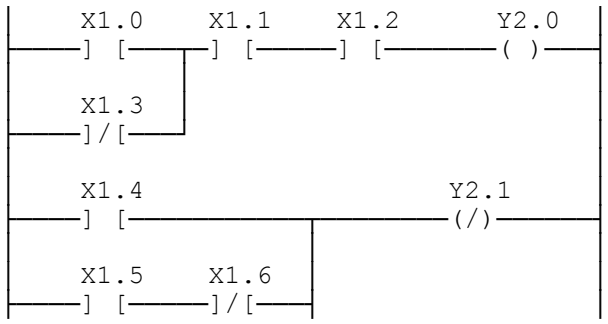



```
X1.0 AND (    ) AND X1.4 TO Y2.0
   X1.1 OR X1.2 OR X1.3
```

7.3.2 BOOLEAN PROGRAMMING CONTINUED

To facilitate writing more complex Boolean expressions, up to eight levels of parentheses may be used. Nested opening brackets are not permitted, but up to 8 nested closing brackets are allowed.

7.3.2.1 COMBINATIONAL LOGIC CIRCUITS



The above ladder diagram, illustrates a combinational logic circuit to switch on two Outputs.

The Y2.0 Output will operate when Input X1.0 OR NOT Input X1.3 is true, AND Inputs X1.1 AND X1.2 are true.

The Processor utilizes this 'English' description of a logic function in order to determine an instruction statement. For example: The instruction statement of the second rung TO Output Y2.1 is expressed as:

X1.4 OR (X1.5 AND NOT X1.6) TO NOT Y2.1

This is a Boolean language statement used to represent a combinational logic circuit or relay ladder diagram.

7.3.3 DESIGNING A BOOLEAN LOGIC PROGRAM

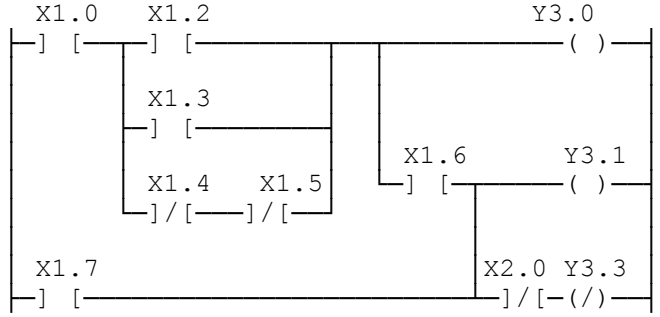
The section covers the design of an applications program using the Boolean language. The Boolean language consists of the following words:

AND, OR, NOT, TO, ON, OFF

These words are used with labels representing Inputs, Outputs, flags and timer counters to mimic a relay ladder or logic diagram circuit. The first stage in program design is to design the relay ladder diagram or combinational logic diagram.

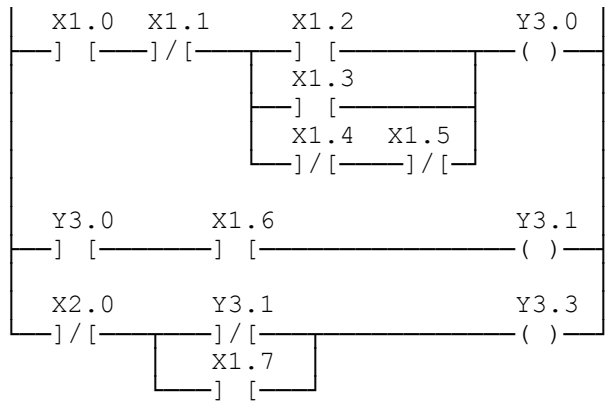
The relay ladder diagram is usually developed from an electrical circuit which the Processor is to emulate. In the **Series 100** Processor there is not really any restriction on the complexity of the relay ladder diagrams rungs, although an attempt at this stage to simplify the design may well make the interpretation into Boolean much easier.

7.3.3 DESIGNING A BOOLEAN PROGRAM CONTINUED



X1.0 AND (X1.2 OR X1.3 OR (NOT X1.4 AND NOT X1.5)) TO Y3.0 AND X1.6 OR X1.7 TO Y3.1 AND NOT X2.0 TO NOT Y3.3

The above Boolean expression is the solution to the above relay ladder diagram, while it shows the flexibility of the **Series 100's** Boolean language, it no doubt took quite some thought to work out the expression. By trying to simplify the diagram first, perhaps using flags, the Boolean conversion becomes much simpler.



X1.0 AND NOT X1.1 AND (X1.2 OR X1.3 OR (NOT X1.4 AND NOT X1.5)) TO Y3.0 Y3.0 AND X1.6 TO Y3.1 NOT X2.0 AND (X1.7 OR Y3.1) TO NOT Y3.3

7.3.4 PROGRAMMING WITH BOOLEAN LOGIC

To insert the Boolean expression as a program instruction, a line number must be use to identify the instruction statement. If the following Boolean expressions are to be inserted:

X1.0 AND NOT X1.1 TO Y2.0 X1.2 OR NOT Y2.0 TO Y2.1

Simply prefix the Boolean equation by a line number in the range 0 to 65,534.

For instance:
10 X1.0 AND NOT X1.1 TO Y2.0 (cr) 20 X1.2 OR NOT Y2.0 TO Y2.1 (cr)

7.3.4 PROGRAMMING WITH BOOLEAN LOGIC CONTINUED

A (cr) denotes depressing the carriage return key on the programming Terminal. The Editor commands can be used to modify or examine the entered instructions.

7.3.5 USING VARIABLES IN BOOLEAN LOGIC

Boolean programs consist of logic functions only. However, there are occasionally requirements to use variable manipulation in a Boolean program. For example, loading the Set Point of a timer/counter.

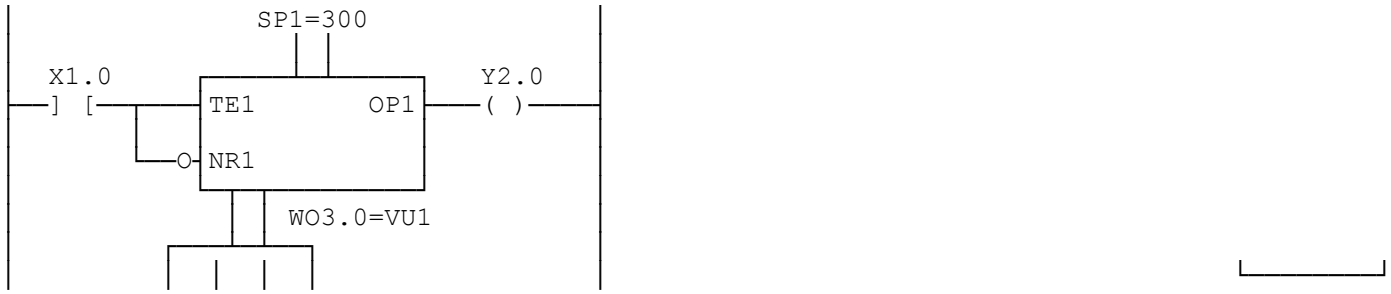
To assign a variable to a SP, the = character is used.

```
A = # ( 1 — 9999)
SP10 = A
```

To assign Set Point of timer/counter 13 = to Set Point of timer/counter 10, use:

```
SP13 = SP10
```

The character, = , is used for Timer Set Point assignment. A delay ON timer is required with a Set Point of 30 seconds.



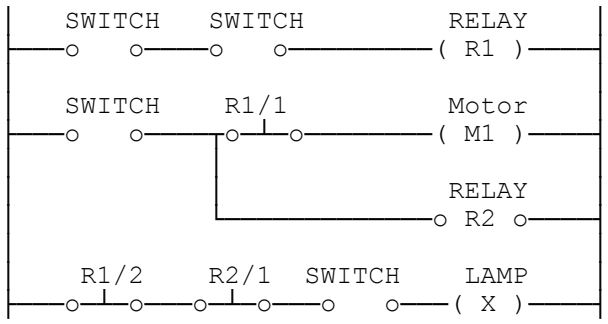
The corresponding applications program would be:

```
10  SP1 = 300
20  X1.0 TO TE1 TO NR1
30  OP1 TO Y2.0
40  WO3.0 = VU1
```

7.3.6 FLAGS (Internal Relays)

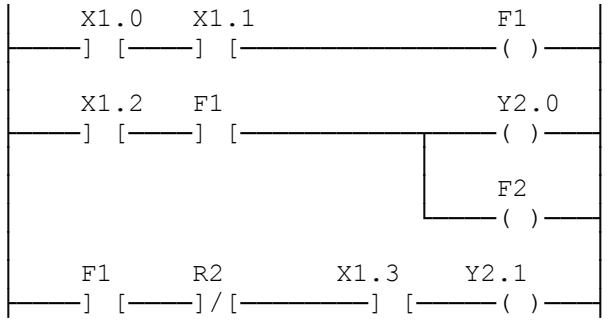
The Processor has 256 internal locations, often referred to as Internal Relays for storing the results of logical operations that do not need to interface directly with I/O lines. These Flags are designated F0 to F255 and are used in the same manner as latching relays or conventional flip flops. Flags occupying the designations F0 to F127 are always cleared on power—up while F128 to F255 retains status at power—down.

The basic idea of using a Flag, is from the traditional design of hard wired relay logic, electrical control panels, where multi—contact internal relays were used to create the combinational logic networks.



Typical Relay Logic Circuit

In the Processor, these internal relays (Flags) are represented by a label, Fn. Where n is a number between 0 and 255 representing 256 independent Flags. If the above relay logic circuit were replaced by a **Series 100** with the switches connected to Input Modules and the motor and lamp being driven by Output Modules. The circuit diagram now becomes a relay ladder diagram.

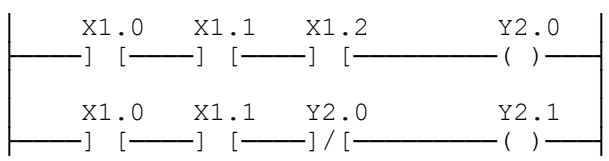


7.3.6 FLAGS CONTINUED

F1 and F2 are flags representing the internal relays. The relay ladder diagram can now be converted into Boolean statements.

```
X1.0 AND X1.1 TO F1
X1.2 AND F1 TO Y2.0 AND F2
F1 AND NOT F2 AND X1.3 TO Y2.1
```

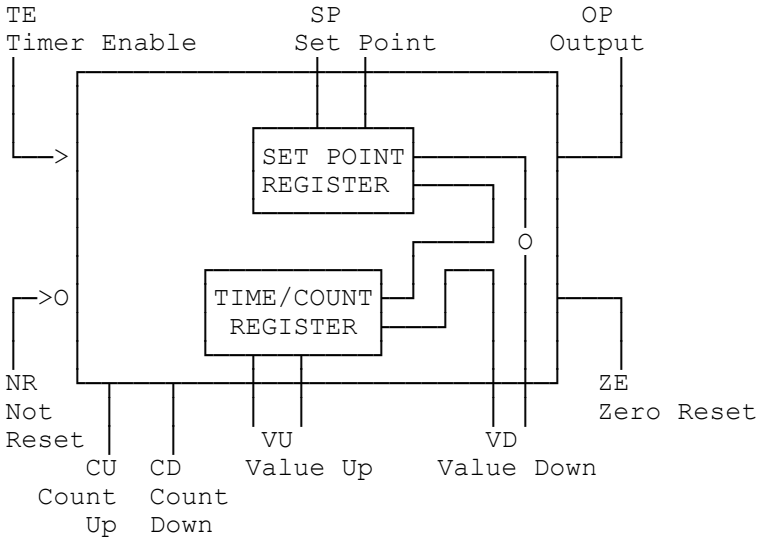
The use of flags allows for simplicity in design, in particular, if an existing electrical circuit with internal relays is replaced by a programable controller. However, the **Series 100** programming language is versatile enough to omit flags from a design. The previous ladder diagram could have been replaced by:



X1.0 AND X1.1 AND X1.2 TO Y2.0 X1.0 AND X1.1 AND NOT Y2.0 TO Y2.1

7.3.7 TIMERS AND COUNTERS

Processor has 63 Timer/Counters and 1 high speed counter. Timer/Counters can either be used as Timers with a timing range of 0.1 to 999.9 seconds, or as Counters with a counting range from 0 to 9999, depending on how the are connected by the applications program.



Block Diagram of a Timer/Counter

Each Timer/Counter has three variables, four discreet Inputs and two discreet Outputs. The variables are the Set Point (SP), Value Up (VU) and Value Down (VD). The discreet Inputs are, Timer Enable (TE), Not Reset (NR), Count Up (CU) and Count Down (CD), and the discreet Outputs are Output (OP) and Zero (ZE).

7.3.7.1 TIMER/COUNTER VARIABLES

These have three separate registers, namely:

VU - Value up: contains the current Timer or Counter value.

VD - Value down: contains the time remaining or count to elapse before the Set Point is reached.

SP - Set Point: contains the time or count when an output will occur.

The values of VU and SP can be preset from within a program.

SP15 = 30 VU10 = 47

It should be noted that the VU register cannot be incremented beyond the SP value or, below zero.

7.3.7.2 TIMER/COUNTER INPUTS

TE (Timer Enable) - When TE is energized Timer/Counter behaves as a Timer and is internally incremented by a pulse to VU and VD registers every 100msec (0.1 sec). When TE is not energized, the Timer/Counter responds to NR, CU and CD Inputs.

NR (Reset when de-energized) - resets VU register to zero and, VD register to Set Point value.

CU (Count Up) - increments VU register, decrements VD register for every pulse or event.

CD (Count Down) - decrements VU register, increments VD register for every pulse or event.

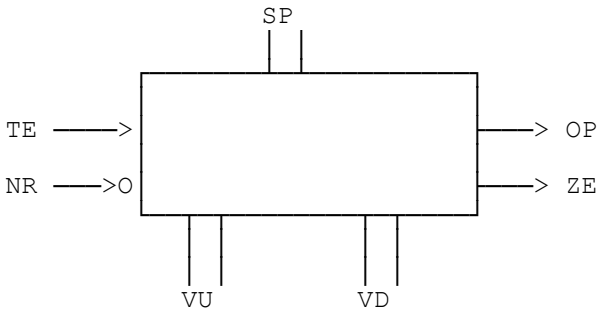
7.3.7.3 TIMER/COUNTER OUTPUTS

These can be used within a program:

OP (Output) - is energized when VU = SP or VD = zero.

ZE (Zero Output) - is energized when VU = zero or VD = SP.

7.3.7.4 USE AS A TIMER



To operate as a timer set the timing range. This is set by SP register which can be loaded with a number between 0 and 9999.

The NR Input is used to clear the timing register, when NR is at logic 0 it clears the register, when NR is at logic 1 it is Not Reset and timing may begin.

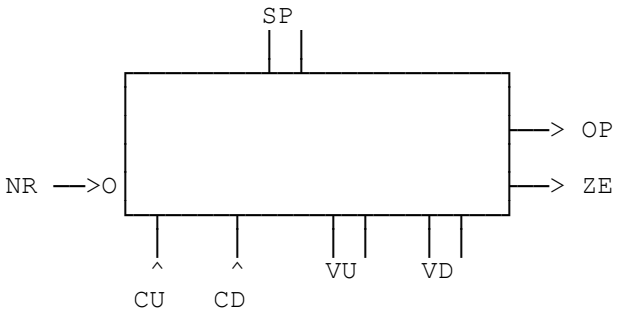
Operation of TE Input going to a logic 1, causes the timing register to increment by one every 0.1 seconds. When number in timing register is equal to SP, OP is set to logic 1.

Thus if SP = 100, and TE goes to logic 1, it will take 100 x 0.1 sec, or 10 sec, before OP is set high. The timing range is 999.9 seconds.

The ZE Output is set to logic one, when VU = 0.

VU is current value of timing register, which represents elapsed time, and VD is the difference between SP and VU and hence represents time to reach SP.

7.3.7.5 USE AS A COUNTER



In counting mode, the Timer/Counter does not use Timer Enable, TE Input. It uses two other discreet Inputs, Count Up (CU) and Count Down (CD).

CU - increments Register every time this Input changes state from logic 0 to logic 1.

CD - decrements Register every time this Input changes state from logic 0 to logic 1.

If Set Point was loaded with 20, then Output would be set high when number of pulses on CU is 20 more times than pulses on CD.

7.3.7.6 CONNECTING AS A TIMER IN THE APPLICATION PROGRAM

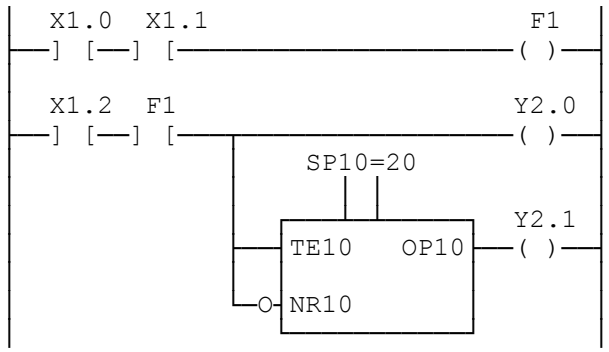
Addressing of particular timer/counter functions is shown by the following examples:

TE1 Represents Timer Enable Input of Timer/Counter 1.

NR20 Represents Not Reset Input of Timer/Counter 20.

CU18 Represents Count Up Input of Timer/Counter 18.

By using labels as part of applications program, timer/counters can be connected in the relay diagram.



7.3.7.6 CONNECTING A TIMER CONTINUED

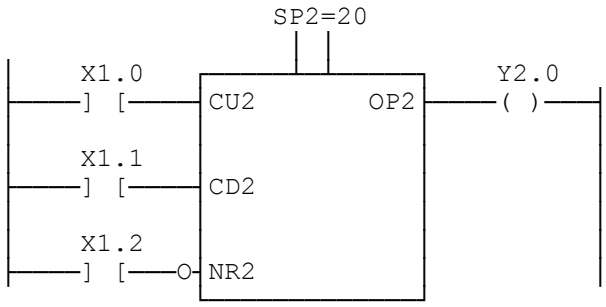
The Boolean expression is:

```
SP10 = 20
X1.0 AND X1.1 TO F1
X1.2 AND F1 TO Y2.0 TO TE10 TO NR10
OP10 TO Y2.1
```

When X1.2 and F1 are zero, TE10 and NR10 are zero and timer is reset and not enabled.

When X1.2 and F1 are at logic one, TE10 is enabled and NR10 is not reset, allowing timer/counter register to increment every 0.1 seconds until it reaches 20. Set point value, OP10 goes high and Y2.1 is turned ON.

This is a 2-second delay on timer connected as a counter:



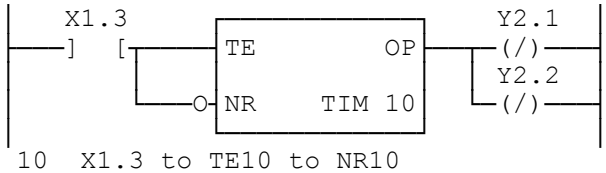
In the above ladder diagram, timer/counter 2 is used as a differential counter. When X1.0 has changed state 20 more times than X1.1, then Y2.0 will be turned ON.

Similar to discreet Inputs and Outputs, X1.1 and Y2.3, flags F21, are discreet Inputs and Outputs of a Timer/Counter. They can be used in Boolean equations which represent rungs of a relay ladder diagram.

7.3.7.7 TIMER EXAMPLES

The following timer configurations illustrate the Ladder and Boolean definitions.

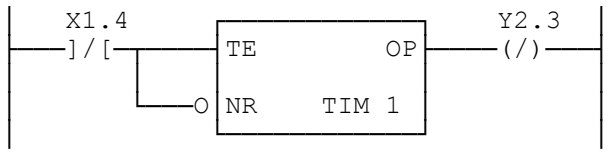
Delay On Timer - Reset when X1.3 is de-energized.




```
10  X1.3 to TE10 to NR10
20  OP10 to Y2.1
30  NOT OP10 to Y2.2
```

7.3.7.7 TIMER EXAMPLES CONTINUED

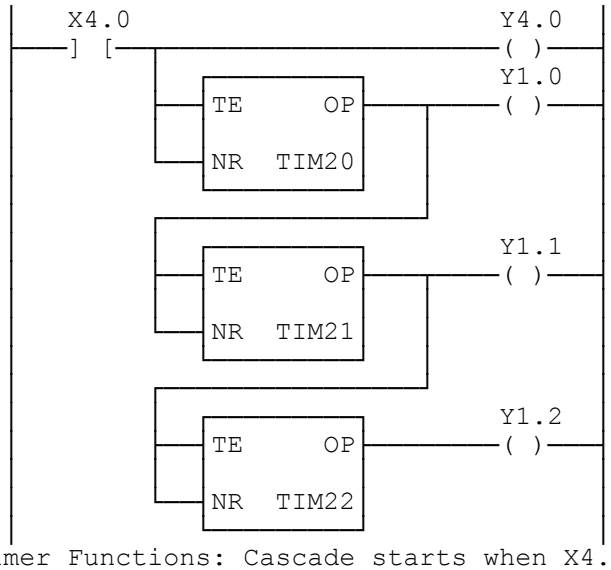
DELAY OFF TIMER - Reset when X1.4 is energized.




```
40  NOT X1.4 to TE11 to NR11
50  NOT OP11 to Y2.3
```

To program the Set Point: 60 SP10 = 25

Cascaded Delay On Timer With 'RUN'Indicator - Reset when power is removed.



Timer Functions: Cascade starts when X4.0 is made, Y4.0 is also energized.

```
60 X4.0 TO TE20 TO NR20 TO Y4.0
70 OP20 TO TE21 TO NR21 TO Y1.0
80 OP21 TO TE22 TO NR22 TO Y1.1
90 OP22 TO Y1.2
```

Opening X4.0 will cause timers to reset.

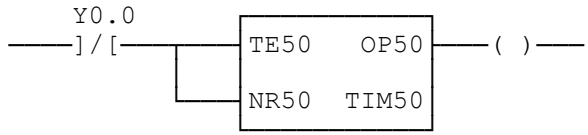
CYCLE TIMER 1 - To make the above into a cycle timer modify line 60 to read:

```
60  X4.0  AND  NOT  Y1.2 TO TE20 TO NR20 TO Y4.0
```

CYCLE TIMER 2 - A very simple cycle timer giving a 10 ms. pulse is:

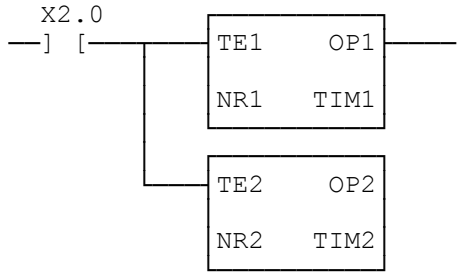
```
100 NOT Y0.0 TO TE50 TO NR50
110 OP50 TO Y0.0
```

7.3.7.7 TIMER EXAMPLES CONTINUED

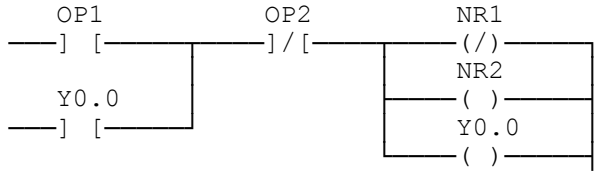


The time between each pulse is adjusted by SP50. If a longer pulse is required, then a second timer is used to control the ON time:

```
100 X2.0 TO TE1 TO TE2
```




```
110 OP1 OR Y0.0 AND NOT OP2 TO Y0.0  TO NOT NR1 TO NR2
```

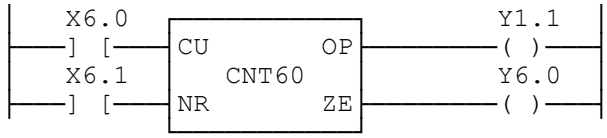


X2.0 enables 2 timers. Both the ON time and the OFF time are individually adjustable.

7.3.7.8 COUNTER EXAMPLES

The following counter configurations illustrate ladder and Boolean methods.

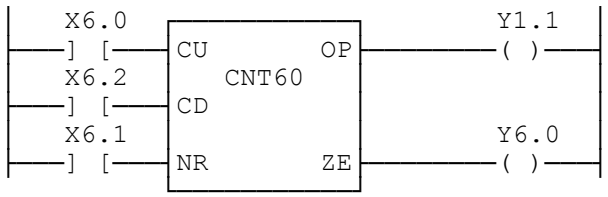
Up Counter - Count when X6.0 closes. Y6.0 energized when count=0.
Y1.1 energized when count = SP
X6.1 energized to enable the counter.




```
200 X6.0 TO CU60
210 X6.1 TO NR60
220 OP60 TO Y1.1
230 ZE60 TO Y6.0
```

7.3.7.8 COUNTER EXAMPLES CONTINUED

Up/Down Counter - Expanding the above example to include a down counter involves just adding one line:

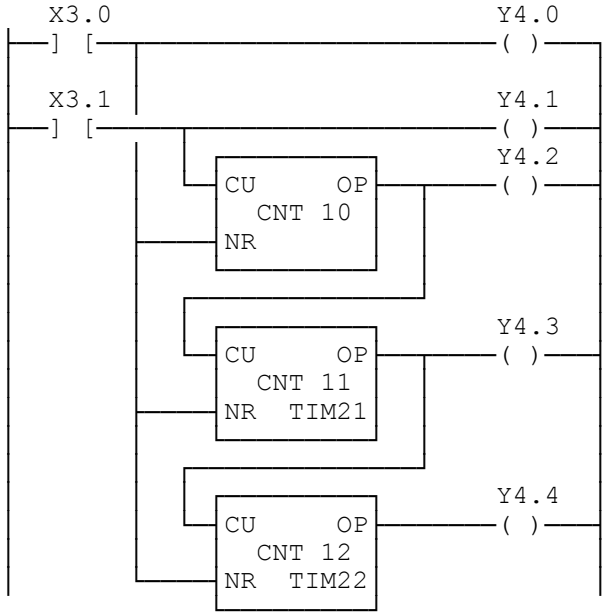


240 X6.2 TO CD60

Note: The CD will never decrement below zero.

CASCADED COUNTERS WITH POWER RESET

Reset when X3.0 is closed.
Count every time X3.1 closes.
Y4.1 indicates CU pulse.



100 X3.0 TO NR10 TO NR11 TO NR12 TO Y4.0
110 X3.1 TO CU10 TO Y4.1
120 X3.1 AND OP10 TO CU11 TO Y4.2
130 X3.1 AND OP11 TO CU12 TO Y4.3
140 X3.1 AND OP12 TO Y4.4

Counter 0 — A special sixty—fourth timer/counter location is available for use as a high speed counter, designated counter zero.

7.3.7.8 COUNTER EXAMPLES CONTINUED

Count Up Input is connected directly to Input X0.0 on the Processor and enables the detection of pulses up to a frequency of 10Khz.

Counter 0 is configured in hardware for high speed operation. NRO, when de—energized, resets the counter to zero. X0.0 behaves like a CU Input and when VUO = SPO, OPO is set. TEO, CDO, and ZEO do not function, and must not be used. The contents of VDO and VUO are accessible as normal, and can be displayed on the Engineers Panel. Maximum count rate is 10kHz.

OP0 is set when VU0 = SP0

Typical applications of this Input:

— Motor shaft speed
- High speed component counting
- Frequency measurement

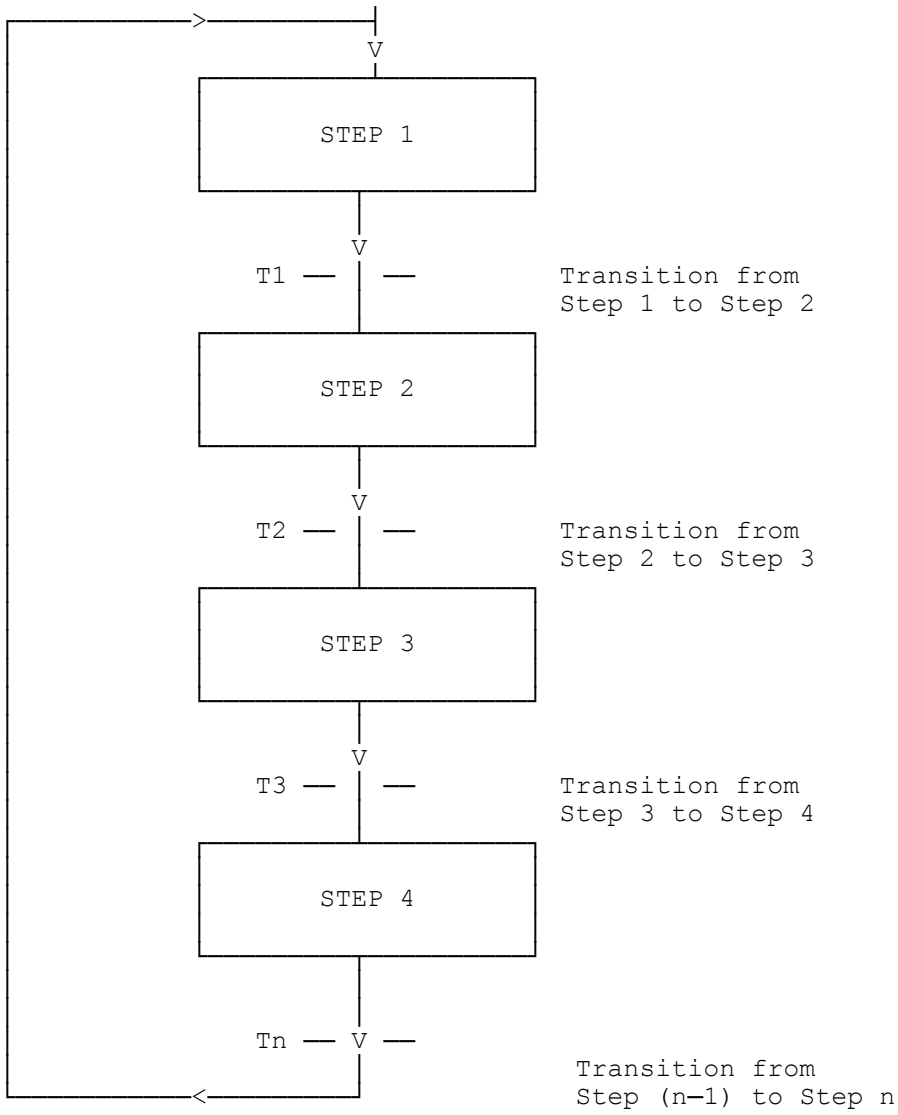
Note: Always use shielded leads for connecting Input line X0.0.

7.4 SEQUENCE PROGRAMS

A Boolean Program can be considered as a parallel program, that is, all the logical functions are executed together. This is not true in all cases since each Boolean instruction is executed in sequence. However, as the update of the I/O Modules is carried out at the end of this sequence and as all Outputs are updated simultaneously, the effect is that of a parallel program.

A sequence program can be considered as a set of consecutive, parallel sub-programs. Each of the sub-programs can be considered as a step or "job to be done". When the "job is done", then a transition occurs which starts the Processor operating on the next step, the "next job to be done".

Diagrammatically a sequence program would appear as:



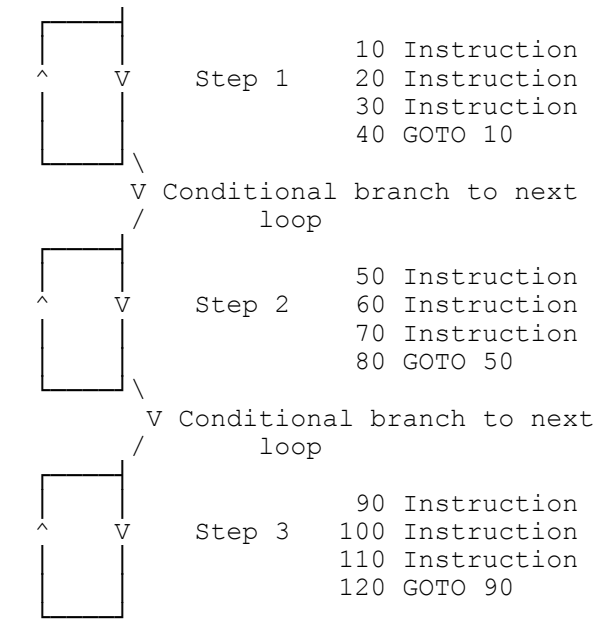
7.4 SEQUENCE PROGRAMS CONTINUED

A new instruction is required to enable the Processor to branch from one loop to another. The instruction is known as a conditional branch instruction and in the **Series 100** language, its format is:

IF condition THEN GOTO line number.

Each step in the sequence program can be considered as an inner loop. The step or inner loop is continuously executed until a condition causes the Processor to branch to another loop.

PROGRAM FLOW

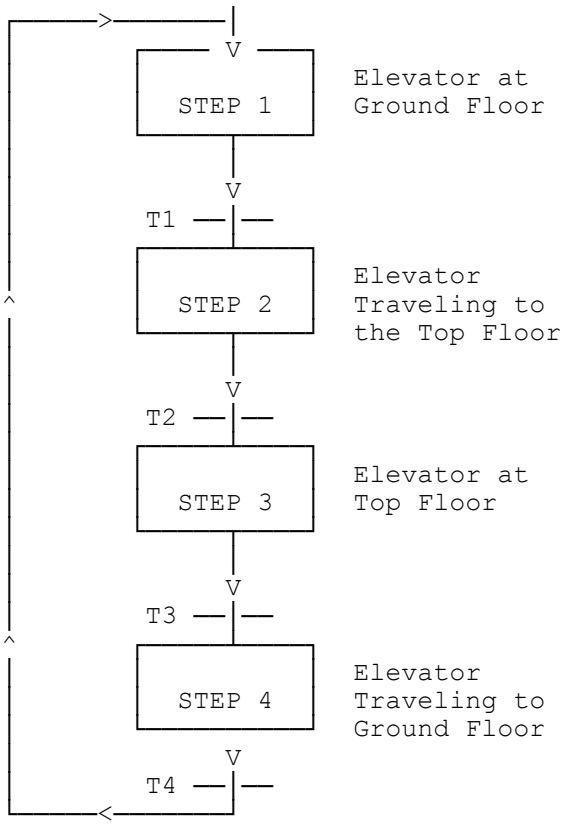


In a sequence program, the instruction controls transition from one step to another. Consider the example:

```
10 X1.0 AND X1.1 TO Y2.0
20 X1.2 OR X1.3 TO Y2.1
30 IF X1.4 AND Y2.1 THEN GOTO 50
40 GOTO 10
50 X1.5 OR X1.3 TO Y2.0
```

The Processor will execute the loop consist of program lines 10 to 40. When Input X1.4 and Output Y2.1 are both simultaneously ON, then line 30 will cause the Processor to branch to line 50, which is the first line of the second loop.

To illustrate the concept of sequence programs consider the example of the control of a simple two story elevator. In this example the control an elevator is broken down into a sequence of four steps.



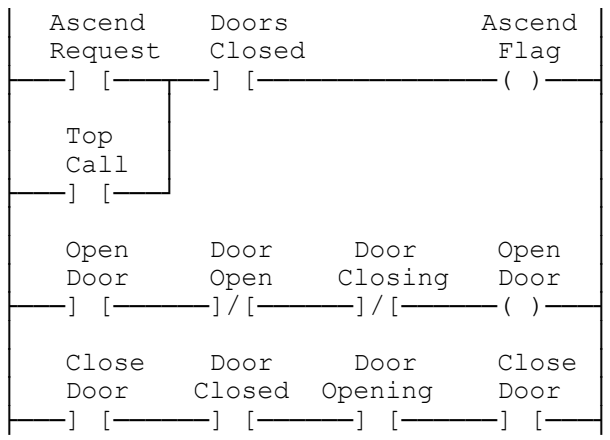
Each step is a sub-program. Step 1's sub-program could be:

Check and control the opening of the elevator ground floor doors.
Check if the internal 'ascend request' button has been pressed.

Check if the top floor 'request elevator' button has been pressed.

7.4 SEQUENCE PROGRAMMING CONTINUED

The relay ladder diagram could be:



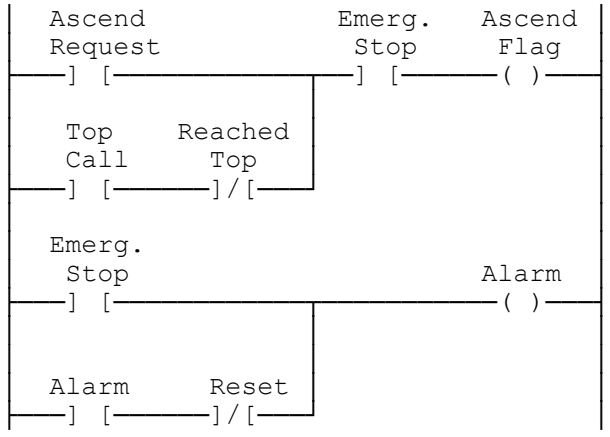
Using Input and Output labels, the Boolean program could be:

```
10 X1.0 OR X1.1 AND X1.2 TO F1
20 X1.3 AND NOT X1.4 AND NOT X1.5 TO Y3.0
30 X1.6 AND NOT X1.7 AND NOT X2.0 TO Y3.1
40 IF F1 THEN GOTO 100
50 GOTO 10
```

Where F1 is a flag used to indicate a request for the elevator to ascend. If F1 is set to logic 1, then the Processor branches to step 2, which controls the ascent of the elevator. Step 2's sub—program could be:

Start the ascend motors.

Check if the emergency stop button has been pressed with a corresponding relay ladder diagram.



The corresponding Boolean program:

```
100 X2.1 OR (Y3.1 AND NOT  X2.2)  AND NOT X2.3 TO Y3.1
110  X2.3  OR  (Y3.2 AND NOT X2.4) TO Y3.2
120 IF X2.2 THEN GOTO 200
130 GOTO 100
```

Input X2.2 is a limit switch which indicates the elevator has reached the top floor and causes the transition to step 3. Steps 3 and 4 can also be coded into relevant Boolean sub-programs.

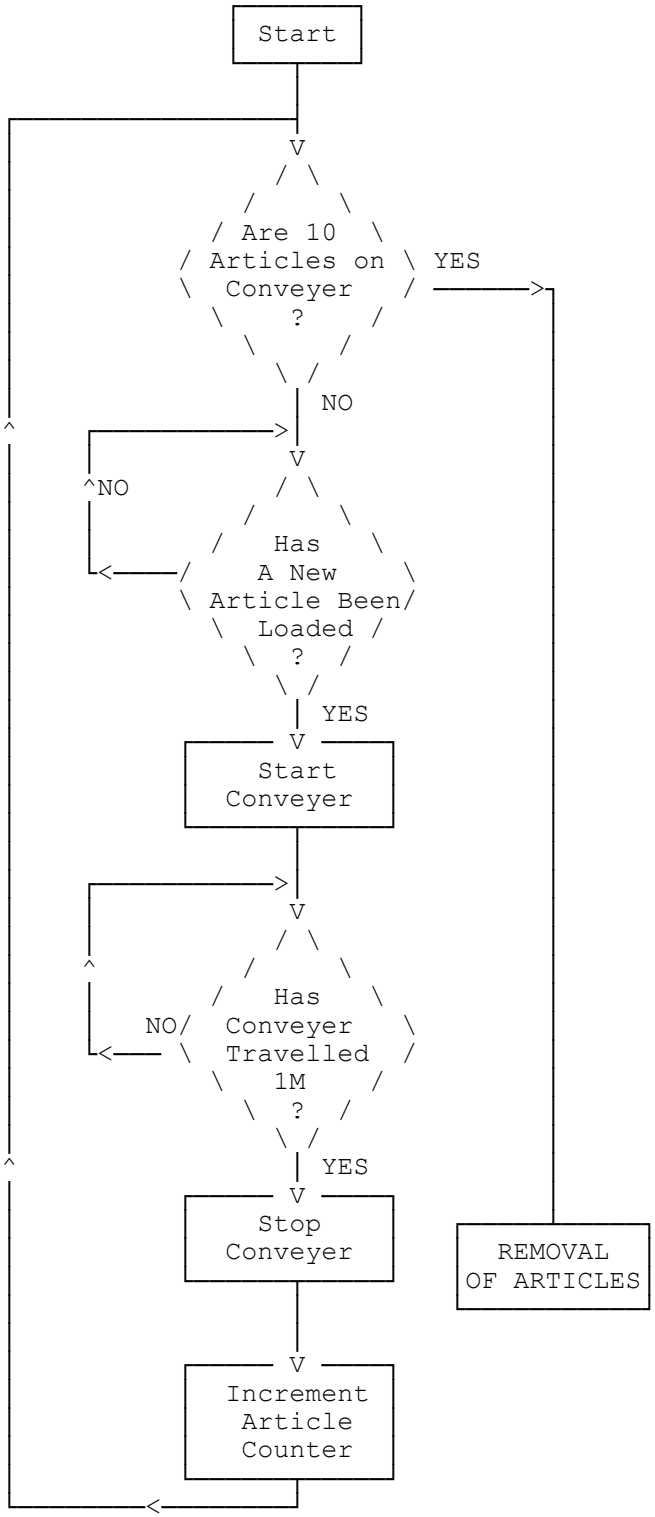
7.5 FLOW CHARTS

Flow charts are a useful diagrammatic means of illustrating various paths through a complex sequence program.

In a flow chart, a diamond represents a conditional branch If the condition is true, flow will take one direction, if the condition is false, the flow will take an alternative direction.

As an example, consider a packaging process, where a conveyor is loaded with various articles. Every time an article is placed onto the front of the conveyor, the conveyor is started and moves until a distance of 1 meter has been reached. This continues until 10 articles have been placed on the conveyor. Then a loading arm is lowered to remove all the articles. A flow chart of the sequence might be:

7.5 FLOW CHART CONTINUED



7.5 FLOW CHART CONTINUED

A **Series 100** program for the above packaging system might look like:

```
10  SP1 = 10                    ! Set Set Point to 10
100 IF OP1 THEN GOTO 180        ! Check article counter Output
110 IF NOT X1.0 THEN GOTO 110   ! Has proximity detector has sensed an article
120 ON Y2.0                     ! Turn on the conveyor
130 IF NOT X1.1 THEN GOTO 130   ! Check if a 1 meter travel sensor has been set
140 OFF Y2.0                    ! Turn off conveyor
150 ON CU1
160 OFF CU1                     ! Pulse the article counter Input
170 GOTO 100
180 ARTICLES REMOVAL            ! Branch to another subroutine
```

7.6 VARIABLES

Variables are internal memory locations used to temporarily store either numbers, known as numeric variables or ASCII text strings known as string variables.

7.6.1 NUMERIC VARIABLES

A Processor has the capacity to store up to 1365 in an 8K Memory system or 4095 numeric variables in a 16 K system.

This limiting factor, 1365 or 4095, is thus an important number. For example, suppose the Processor was required to periodically store away an analog value representing a liquid's flow measurement. It is important to relate a flow reading to the time of day. Every time a reading is stored, the time of the reading is stored too. Therefore for every reading two variables must be stored - flow and time. With a limit of 1365 variables, the maximum number of readings that can be stored is thus:

```
1365 / 2   =   682
```

With 24 hours per day and 60 minutes per hour, the maximum reading rate would be:

```
(24 x 60) / 682 = 2.11 minutes.
```

In actual fact, all 1365 variables would probably not be available for data logging, as some would be needed for data manipulation such as finding the square root of a measured variable.

As previously stated, there are normally 208 numeric variables and 26 variables names, A to Z, each with 8 elements. In order to increase the number of numeric variables, the BASIC statement DIMENSION (shortened to DIM) must be used.

7.6.2 NUMERIC AND VARIABLE DATA MANIPULATION

The Processor uses a type of data manipulation called OPERATORS. The data can be either numeric or alphanumeric strings.

7.6.2.1 MATHEMATIC OPERATORS

Operators define arithmetic operation to be performed and include add, subtract, multiply and divide. Relational operators such as less than, equal to are included.

Arithmetic Operators with their program symbol are shown below:

```
  +  addition          ()   parentheses        FRAC
  -  subtraction      ABS   Absolute value     *  multiplication
 INTEG Integer value   /  division            SQR   Square root
```

7.6.2.1 MATHEMATIC OPERATORS CONTINUED

The **Series 100** always evaluates mathematical functions in the order encountered in a program line just as it does for Boolean operations. Care must be taken when defining a mathematical equation or function to insure that it functions as expected.

Parenthesis are essential for grouping and separating related and non-related parts of a mathematical function. In the example A = 5 * 3 - 2/3 = 4.333. This is different than A = 5 * (3 - 2)/3 = 1.667. This is due to the location of the parenthesis.

ABS - returns the absolute value of an expression, it ignores negative signs so the result is always plus.

```
ABS - 3.24 = 3.24

10  A = ABS B  Where B contains the value -3.24, A will
    contain the result (+) 3.24.
```

FRAC - returns the fractional portion of a value to the right of the decimal point. The integeris discarded. Note that some rounding of the fractional value may occur.

INTEG - returns the integer value of an expression, any fractional part is discarded.

```
INTEG 3.24 = 3 10 A = INTEG B
Where B contains the value 3.24, A will contain the result 3.
```

Although negative numbers may be used, be aware that the result is always rounded down.

```
INTEG - 3.24 = -3     10 A = INTEG B
Where B contains the value -3.24, A will contain the result=3.
```

SQR - returns the square root of an expression. The result is always positive and therefore negative numbers will generate an error.

```
SQR 4 = 2
SQR 0.25 = 0.5
20 A = SQR B   Where B contains the value 4, A will contain
               the result 2.
```

Relational Operators allow the evaluation of conditional statements.

```
LT  < than            NE  not =
LE  < than or =       GT  > than
EQ  =                 GE  > than or =
```

Used in conjunction with the Basic statements IF, THEN, this example shows how operators are used:

```
120 IF A EQ 10 THEN GOTO 190
```

7.6.3 NUMERIC DATA INPUT AND OUTPUT OPERATORS

Other forms of data manipulations are available within the **Series 100** for handling the various I/O available, using Byte, Word and Module for Digital I/O and Analog Modules for Analog I/O.

For standard BCD interface, (100-3-9030 Thumbwheel Switches or 100-3-9031 Seven Segment Displays, the WI and WO or BI and BO operators are used.

WImm.0 ,Word In, gets the information from the Thumbwheel and **WO**mm.0, Word Out, outputs four BCD digits and manipulates them as variables. Numbers less than one will be output as zero, and numbers greater than 9999 will be output as 9999.

BImm.0, Byte In gets the Thumbwheel information and **BO**mm.0 Byte Out outputs two BCD digits and manipulates them as variables. Numbers less than one will be output as zero, and numbers greater than 99 will be output as 99.

For binary interface use MI/MO.

MImm.0, Module In, gets the Thumbwheel information and **MO**mm.0, Module Out, outputs, in binary form and manipulates this data as variables. Numbers less than one will be output as zero, and numbers above 255 will be output as 255.

The above three operators may be used freely in data manipulations, with the exception that they may not be used in the second part of a relational expression.

```
120 IF WI12.0 EQ 1234 THEN ON Y2.7       Is acceptable

130 IF 1234 EQ WI12.0 THEN ON Y2.7       Is not acceptable
```

7.6.4 ANALOG INPUT AND OUTPUT OPERATORS

For Analog module interface use the AINmm.n and AOTmm operators:

AINmm.N takes the current value of channel N of the Analog Input module at locations mm and mm+1. It may only be used to place the Input value in a variable; no other direct manipulation of AIN is possible. The AIN statement should sample the Analog Input module at a rate faster than module conversion rate to avoid missed samples. For the 100-3-9041-08 Single Channel Module, this is every 80msec, for the four channel modules this is 320msec. The input number lies in the range ±4095.

```
120 A(0) = AIN5.0 130 A(1) = AIN5.1
```

AOTmm is used to output data to up to four channels in sequence of an Analog Output Module such as 100-3-9042-XX. The number range lies between 0 and 4095. Numbers less than one will be output as zero, numbers above 4095 will be output as 4095. Following is the line for a Single Channel:

```
120 AOT12 = A(5)
```

7.6.4 ANALOG INPUT AND OUTPUT OPERATORS CONTINUED

For a Two Channel:

```
120 AOT12 = A(5), B(6,7)
```

And for a Four Channel:

```
120 AOT12 = A(5), B(6,7), 1234, Z
```

This places variable element A(5) in channel 0, and B(6,7) in channel 1.

Note: **AOT** is only executed from Task "0".

Note: for AIN and AOT, full scale Input or Output occurs for a count of 4000, and thus the max number 4095 allows for some small amount of over-ranging in the software.

7.6.5 REAL TIME CLOCK

Certain special function registers within a **Series 100** system may be read as if they were variables. Although some may only be read and not written. These registers will include:

TIME - holds time of day as HHMM.
DAY - holds day of week as 1 to 7.
DATE - holds day and month as DDMM.

In addition, three registers, OFFTIME, OFFDAY, OFFDATE are loaded with TIME, DAY, and DATE on power down.

All the registers may be manipulated as read only variables. However, the Time, Day and Date registers can be initialized via the Engineers Panel to set the current time and date.

To access hours and minutes, the following could be used:

```
10 A = TIME
20 B = INTEG A
30 C = B/100
40 D = INTEG C ! Hours
50 E = FRAC  C ! Minutes
```

Care should be taken to correctly condition results from the real time clock because of the way the internal storage operates. It is advisable to use the INTEG function to condition the parts prior to comparison with a fixed time when sequencing. When reading these registers into variables, minor differences in the routines used to calculate the numbers can mean differences of one or two in the least significant bits in the internal storage and an equality test may fail.

7.7 BASIC PROGRAMMING LANGUAGE

BASIC is a high level programming language designed to allow programs to be generated using the English language. The version of BASIC used by the **Series 100** has been structured for high speed execution and is an ideal media for the solution of complex applications.

Basic commands can be freely mixed with Boolean statements which together form a very powerful and comprehensive set of central instructions.

7.7.1 BASIC COMMANDS AND STATEMENTS

7.7.1.1 CHR$

To enable escape sequences to be sent to a terminal. Sequences for cursor positioning, and setting out tables are examples of the CHR$ statement acting on ASCII characters. To send an ESC sequence of an ESC and two characters, the following code might be used:

```
10 B  = 27            !Escape
20 A$ = CHR$ B
30 C  = 61            ! X position
40 D$ = CHR$ C
50 E  = 34            ! Y position
60 F$ = CHR$ E
70 G$ = A$ + D$ + F$
80 PRINT G$
```

Note: The CHR$ function operates on variables whose numeric value lies in the range 0 - 127.

Note: Always remember to dimension the array using the 'DIM' statement early in the program.

```
10 DIM A$
```

7.7.1.2 CLEAR

CLEAR zeros all variables including SP, VU and retentive Flags. It may be used prior to machine initialization if retentive data features are not required by program. On execution of a CLEAR statement, all variables are set as numeric, with one dimension of 8 elements only.

7.7.1.3 DATA

DATA prefaces lines of data that are to be read by the READ statement for assignment to variables. The data may be numeric or string constants. Pointer DATA statements located anywhere in a program are maintained to locate the next element to be READ. A string constant may only appear at the end of a DATA statement.

```
10 Data 1,2,3,4,"Hello"
```

7.7.1.4 DIM

DIM is used to define the dimensions of an array.

Dim A (10) Dim B$ (10), 50 - where 10 in the # of B$ locations and 50 is maximum number of characters is B$.

The DIM statement will corrupt existing variable data, and it should be executed prior to any reference to variables. Once a string varible has been defined using the DIM statement, the variable is no longer available for use as a numeric variable until redefined by the DIM statement.

7.7.1.4.1 SINGLE DIMENSION ARRAY

The format of the DIM statement is **DIM A(n)** where:
- A represents variable A
- n represents the number of elements.

In the Processor, A(n) is known as a single dimension array, where n represents a number between 0 and 85. This allows up to 85 numeric variables associated with the variable letter.

As an example: 10 DIM A(85)
If it were executed was executed, the variable space would be:

```
A(0), A(1), . . A(7), A(8), . . A(84)
B(0), B(1), . . B(7)
 ,                 ,
Z(0), Z(1),       Z(7)
```

85 (Variable A) + (25 x 8) (Variables B through Z) = 285 variables. If the above instruction was repeated for variable B,

```
20 DIM B(85)
```

The number of variables becomes (2 x 85) (Variables A & B + (24 x 8) (Variables C thru Z) = 362

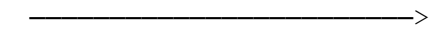
If this was done for all the numeric variable names, A to Z, a total of 26 x 85 = 2,210 numeric variables would be allocated. However, the maximum number of numeric variables the Processor can store is 1365 (8K of Variable Memory). Therefore, the above is an illegal number, and would cause a VAR FAIL, when the Processor entered RUN mode. To summarize:

```
10 DIM A(50) 20 DIM B(3) 30 DIM C(85)
```

These are valid instructions. However if the total number of variables created is greater than 1365, for 8K of Variable Memory, then a VAR FAIL will occur in RUN mode. Remember, the other variables that have not been dimensioned. In the above case, the DIM statemen has created using A, B, C and D, 50 + 3 + 85 + 21 = 159 variables, but there are still 22 variables (E to Z) with the default of 8 elements. Thus the total number is 159 + (22 x 8) = 335 variables.

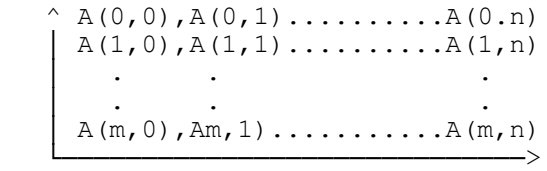
7.7.1.4.2 MULTIDIMENSIONAL ARRAYS

So far, the DIM, statement has been used to create single dimension arrays, **DIM A(85)** creates :



This a single dimension array.

Multidimensional, up to three axes, arrays are also possible. A two dimensional array would be:



This is an m by n, two dimensional array using the variable A.

The two dimensional array is created by the **DIM A(m,n)** statement. For example, the instruction **10 DIM A(4,5)** would create an array

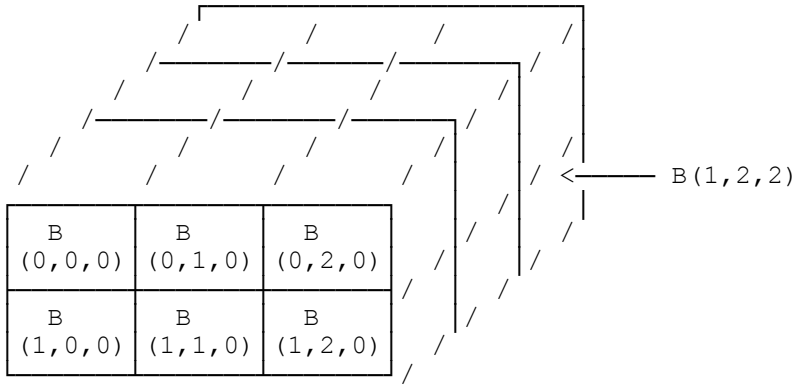
A(0,0)	A(0,1)		A(0,4)
A(3,0)			A(3,4)

which creates 4 x 5 = 20 variables associated with the variable name A.

Similarly a 3 dimensional array:

20 DIM B (2,3,4) would create:

7.7.1.4.2 MULTIDIMENSIONAL ARRAYS CONTINUED



This creates 2 x 3 x 4 = 24 numeric variables associated with the variable name B.

The Processor therefore can store numeric variables as one, two or three dimension arrays. The maximum number of elements in any dimension is 85. Theoretically, DIM A(85,85,85) is feasible. However this would create 85 x 85 x 85 = 614,125 variables, which of course, is much larger than the 1365 maximum (8K of Variable Memory). As an example:

```
DIM A (85,2,4)
DIM B (1,20,9)
DIM C (20,3) DIM D (1)
```

These are all valid statements, be sure the maximum number created is not greater than 1365 (8K of Variable Memory). The above creates (85 x 2 x 4) + (1 x 20 x 9) + (20 x 3) + 1 = 921 variables and variables E to Z have a default of 8 elements, creating a total of 921 + (22 x 8) = 1097 variables, within the 1365 limit.

7.7.1.5 DISPLAY

DISPLAY is identical to the PRINT command but the output will appear on the Engineers Panel and the subsequent semi—colon is ineffective.

7.7.1.6 FOR/NEXT/STEP

FOR v=x1 TO x2 (**STEP** x3)
NEXT v

This command allows repeated execution of a group of statements. When x1, x2, and x3 are evaluated, x1 is loaded to variable v as an initial value, x2 is saved as the final value, and x3 is saved as the increment. NEXT v is placed at the bottom of the loop and when executed, increments variable v by the STEP value; if the result is less then the final value, execution will loop back to the line following the FOR-TO statement. If STEP is omitted, the step value defaults to 1. TIME variables and counter SP, VU and VD may be used as start, end and steps. The limits for x1, x2 and x3 are -32768 to 32767; any (numeric variable) which evaluates to a number outside this range will cause a FOR FAIL.

7.7.1.6 FOR/NEXT STEP CONTINUED

The FOR-TO-NEXT function is a type of loop operation which causes the program to cycle around a loop until the defined limit has been reached. Caution must be exercised when using multiple FOR-TO-NEXT loops within themselves is termed "Nesting". A FOR-TO-NEXT loop may be located within another FOR-TO-NEXT loop
(nested) provided each of these loops is completely isolated from each other. Nesting of FOR-TO-NEXT loops relies on each loop being able to execute its operation with its own 2 -1 FOR-TO-NEXT values and variables. Should one loop cross over into another loop the program will execute using the wrong FOR-TO-NEXT variable. FOR-TO-NEXT statements may be nested up to four levels deep, meaning up to four of these loops may be contained within each other.

The following program shows a single FOR-TO-NEXT example:

```
160 A=1
170 B=14
180 FOR X = A TO B STEP 3
190 PRINT X
200 NEXT X
```

When the RUN command is executed the display will indicate:

```
1 4 7 10 13
```

7.7.1.7 GOSUB (line number)

GOSUB causes the program to save the next line number and unconditionally branch to a subroutine. At the end of the subroutine a RETURN command will cause the program to return to the next line number that has been saved. Subroutines may be nested up to a eight levels deep.

```
110 IF A GT B THEN GOSUB 220
```

This instruction directs program to go to subroutine 220.

```
120 A = B
```

This instruction directs program to return to next Line Number from sub-routine.

Execution of GOSUB will cause a Break as in GOTO.

7.7.1.8 GOTO (line number)

GOTO causes program execution to unconditionally branch to the stated line number. If the stated line number does not exist, the program will not run, and the Editor will display an error message and the source line in error. GOTO must always be at the end of a line, with the exception of the comments.

```
100 GOTO 200 !JUMP TO LINE 200
```

Causes a jump the program to line 200.

7.7.1.8 GOTO CONTINUED

Execution of GOTO causes the current task to relinquish control, or Break, to the next highest priority task. If used in Task 0, GOTO will cause a Break, and Task 0 will only then be resumed after the next I/O update, which may be up to 10msec delay.

It is especially useful to use GOTO statements to keep the program from entering into unwanted areas of sub-routines, or to insure the program does move into a sub-routine as a result of some decision. Where sub-routines are included in a program they should be preceded by a GOTO command to prevent the program execution proceeding into the sub-routine.

7.7.1.9 **IF** (expression true) **THEN**

This command evaluates the expression, and if true executes the remainder of the line following THEN. If the expression evaluates false, execution will continue at the start of the next line. Either BASIC or Boolean statements are allowed as the expression or as the remainder of the line.

```
160 IF A+B GE C THEN GOTO 110
```

This line will cause the program to jump to line 110 IF the sum of A and B is greater than or equal to C.

```
170 If X.10 = X1.1 THEN GOTO 110
```

This line will cause the program to jump to 110 if X.10 and X1.1 are true.

7.7.1.10 **INPUT** v or v**$**

This command returns a prompt character ( ) and waits for the user to enter data via the serial port. This data is loaded to the specified numeric (or string) variable delimited by a carriage return. If the data entered is not valid for the variable type specified, an error message is displayed and the
prompt character will be re-sent. If entry errors are made prior to completing the line with a carriage return, they may be corrected using the RUBOUT key on the terminal. If an INPUT statement is followed by a semi-colon, a Space character will delimit the INPUT string.

Multiple INPUT segments may not be concatenated on the same line of program. INPUT is a **PART COMPLETE** statement and echoing of INPUT characters may be de-selected under the Utility option on the Engineers Panel.

For example, where a program had a number of choices and the decision as to which choice could only be made by the user then the following arrangements would be used.

```
250 PRINT "ENTER COMMAND"
260 INPUT A$;
270 INPUT B

RUN
ENTER COMMAND
LOAD 5
```

7.7.1.10 INPUT CONTINUED

These program lines when encountered will print on the screen the characters between quotation marks, and wait for a response by the user indicated by a question mark. In the above example the user is expected to enter a series of alphabetic characters (string) and a number, LOAD 5 followed by CR. The program must then be written to consider the response of A$ and B. Note while waiting for the user to enter the requested information the program is halted; also A$ delimits on a space (use of ;) while B delimits on a (cr).

If no data is available, the specified string variable will be cleared.

7.7.1.11 ON

ON turns an Output on immediately.

7.7.1.12 OFF

OFF turns an Output off immediately.

Note: The use of **On** and **Off** commands is effective during set-up of initial conditions early in a program.

7.7.1.13 PRINT

PRINT enables output of data (status, informational, fault statements, to a printer. An absolute string or numeric value:

```
10 PRINT "BOILER TEMPERATURE" or
10 PRINT "1,637.94"
```

The statement between quotation marks will be printed. Spaces inserted will be printed as such if they are between quotation marks.

```
10 PRINT A$
```

It will print whatever A$ was previously defined.

Statements which include a character string and a numeric value which is part of some other variable quantity require the use of a semi-colon which will allow the inclusion of that variable on the same line as the previous character string.

```
210 A = 390
220 PRINT "BOILER TEMPERATURE =";
230 PRINT A;
240 PRINT "DEGREES C"
```

After typing 'RUN' the terminal will display:

```
BOILER TEMPERATURE = 390 DEGREES C
```

Note how the semi-colon suppressed the printing on new lines and how the system automatically added the print statements together.

7.7.1.13 PRINT CONTINUED

It is not possible to concatenate a number of segments to be PRINTed on one line; they must be split up so that only one segment occurs per program line.

```
210 A=390
220 B=-5
230 PRINT "BOILER TEMPERATURE=";
240 PRINT A;
250 PRINT "DEGREES C"
260 PRINT "OUTSIDE TEMPERATURE";
270 PRINT B;
280 PRINT "DEGREES C"

RUN
BOILER TEMPERATURE=390 DEGREES C
OUTSIDE TEMPERATURE=-5 DEGREES C
```

7.7.1.14 READ

READ sequentially assigns values found in associated DATA statements to the specified variables. If more DATA is READ than is available, an OUT OF DATA (DTA FAIL) error occurs. Any type of constant may be READ, provided that the associated variable is appropriate.

7.7.1.15 **RESTORE**(line number)

RESTORE resets the READ pointer to the start of the specified DATA statement in the program to allow data to be re-READ.

```
250 RESTORE 260
260 DATA 10,11,12, -9876, "HELLO"
270 DATA "A"
280 READ A,B,C
290 PRINT A;
300 PRINT B;
310 PRINT C
320 READ D,E$,F$
330 PRINT D;
340 PRINT E$;
350 PRINT F$
360 RESTORE 260
370 READ G
380 PRINT G
```

When the RUN command is entered, the following will be displayed:

```
10 11 12
-9876 HELLO A
10
```

7.7.1.16 RETURN

RETURN returns program control to the line following the last GOSUB statement.

7.7.2 PROGRAM DOCUMENTAION

When writing an applications program, it is a very useful feature to be able to document the program with text comments. In the **Series 100** language, instructions and instruction lines can be documented by using the '!' character. For example:

```
10 X1.0 AND X1.1 TO Y2.0 !
```

This allows comments to be added after an instruction

```
20 !
```

This allows a comment or remark to be added to the program. The comment lines are skipped by the Processor when it is in the Run mode. An example of a documented program:

```
10 ! THIS IS A COMMENT LINE
20 X1.0 TO TE1 !  THIS IS A COMMENT AFTER AN INSTRUCTION
30 ! START INDUCTION MOTOR
40 X1.2 OR X1.3 AND X1.4 TO Y2.0
50 ! X1.2  IS AUXILLARY START SWITCH
60 ! X1.3 IS LOCAL START SWITCH
70 ! X1.4 IS SAFETY INTERLOCK
80 ! Y2.0 IS MOTOR CONTACT OR OUTPUT
```

In the above example only lines 20 and 40 contain program instructions, the rest are program documentation comments. The advantage of program documentation is the explanations of instructions to be included in the program for future reference.

Another use of the program documentation facility is for program de-bugging. Consider the program:

```
10 X1.0 AND X1.1 TO Y2.0
20 X1.1 AND X1.2 TO Y2.1
30 X1.1 OR X1.3 TO Y2.2
```

If you wish to temporarily remove line 20 from the applications program, for instance, during de-bugging, you could DELETE line 20 and re-insert it later, or alternatively be EDITING line 20 and inserting an '!' after the line number:

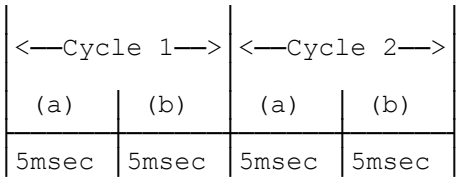
```
20 ! X1.1 AND X1.2 TO Y2.1
```

Line 20 is turned into a comment and thus is ignored in RUN mode. When you wish to re-insert the line, simply re-EDIT line 20 and delete the '!' character.

7.8 PROGRAM EXECUTION AND TIMING

Following RUN command, or power up in RUN mode, the Processor executes each program instruction in numeric order, starting with the lowest line number.

On reaching the last line, the Processor will cease executing applications program and execute several of its own internal programs, for example, updating I/O Modules or checking for internal faults. Having done this, the Processor then repeats cycle, staring by re-executing applications program. This cycle, known as a Processor cycle is controlled to an accurate time interval by internal clock circuits. A normal Processor cycle takes 10 msec to complete and consists of two 5 msec cycles as shown below.

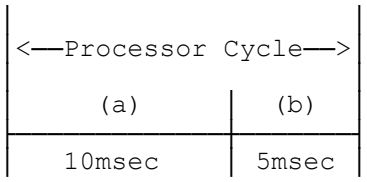


(a) Execute applications program in line number order.

(b) Execute other tasks including update I/O Modules, check for faults.

The I/O scan time is the same as the Processor cycle time, normally 10 msec.

Actual time the applications program takes to complete is a function of the number of instructions and type and length of each instruction. In the above illustration of a 10 msec cycle time, time allocated for execution of applications program is 5 msec. In cases where instruction execution time the is greater than 5 msec, an additional 5 msec is added to cycle time, making the Processor cycle 15 msec.

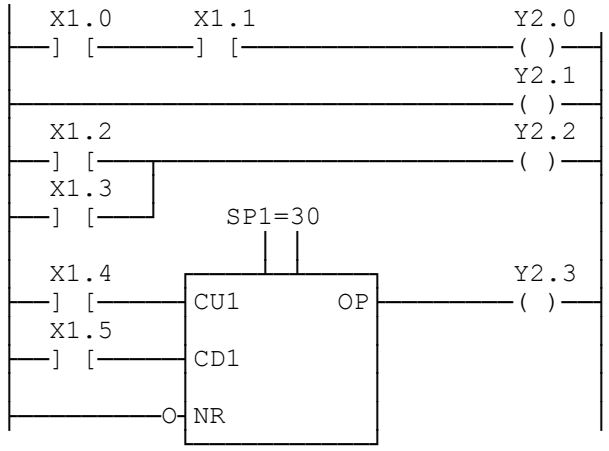


Here, the applications program takes longer than 10 msec then and an additional 5 msec is added to the Processor cycle time. This process continues until the Processor cycle time is sufficient to include the applications program.

As I/O scan time is the same as process cycle time, it too will extend as applications program execution time increases. A shorter execution time yieldsa faster I/O scan rate. For a Boolean program, 5 msec is a long time for enabling 400 Boolean functions to be executed, but is usually sufficient. However, it is a factor that needs to be considered and therefore good practice to attempt minimizing the applications program execution time.

7.8 PROGRAM EXECUTION AND TIMING CONTINUED

Consider the following relay ladder diagram and Boolean program.



The Boolean program is:

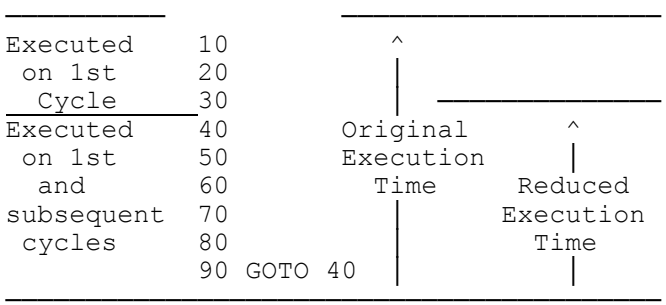
```
10 X1.0 AND X1.1 TO Y2.0
20 ON Y2.1
30 X1.2 OR X1.3 TO Y2.2
40 SP1 = 30
50 X1.4 TO CU1
60 X1.5 TO CD1
70 ON NR1
80 OP1 TO Y2.3
```

Processor executes eight instructions to complete the applications program during each cycle. However, having executed lines 20, 40 and 70 once, their state is set and there is no need to re—execute them again. This fact can be taken advantage of, consider the restructured program below:

```
10 ON Y2.1
20 SP1 = 30
30 ON NR1
40 X1.0 AND X1.1 TO Y2.0
50 X1.2 OR X1.3 TO Y2.2
60 X1.4 TO CU1
70 X1.5 TO CD1
80 OP1 TO Y2.3
90 GOTO 40
```

Using the GOTO instruction creates an inner loop. On executing a GOTO instruction the Processor breaks off execution of applications program and goes on to execute its other tasks in order to complete the Processor cycle. However, at the restart of the Processor cycle, the applications program is executed from the point it was sent to by the GOTO instruction.

Following a power up or RUN command, the Processor enters RUN mode. In the first Processor cycle, applications program starts at the beginning. However, in all subsequent Processor cycles it starts from the position it was sent by the GOTO instruction. The effect is to reduce the applications program execution time.



7.9 MULTITASKING

Most control systems involve a number of related, discrete functions. Some, scanned logic will require frequent attention by the Processor. A multitasking scheduler allows functions or tasks to be undertaken in order of priority so that the frequency of execution of each task may be controlled. Multitasking is a concept which appears to allow the Processor to execute a number functions to simultaneously.

To implement multitasking, the application must be sub-divided into a number of tasks. The task number indicates its priority level, and optionally, a time delay between periods of execution of between 50msec to 2.5 sec, in 10 msec steps. It may be defined as follows:

```
   470 TASK 4 EVERY 100
```

The time delay represents maximum time between parts of a task being executed. Tasks will execute faster than the repeat time where program execution time of various tasks is small compared to the task maximum repeat time.

The default value of task timer is 1 sec. TASK 0 is executed every 10msec unless its duration is longer than 5msec. In this case, 5msec is added to its task timer.

Each enabled task is executed in order of priority to an instruction causing a break, or to an END statement. The exception to this is Task 0. An END statement disables the task, a break saves its status, and the next enabled task is executed. When the lowest priority task has been executed, the highest priority task is the next task.

Every 10msec, the Processor decrements each task timer in descending order of priority. If one has timed out, that task becomes the next task to be executed. The rest of the task timers are not updated. The Task Timer is loaded when task starts or resumes execution or when it times out.

Tasks may occur in any order in the program. The first executable statement in a program is assumed to be TASK 0 if it is not a task header. The program must contain a TASK 0 and two tasks may not have the same Task number.

The status of a task may be viewed using the STATUS instruction. It can be used in an IF/THEN statement. Tasks should be ended with an END.

7.9 MULTITASKING CONTINUED

Eight concurrent tasks can be handled. When the program starts to run, only TASK 0 is enabled. Any other task must be ENABLED by TASK 0 or a previously enabled task.

```
10 ! TASK 0
20
,
60   ENABLE 1  ! Enable Task 1
,
200  END  ! End of Task 0

800 TASK 1 EVERY 200  ! Task 1 Header
 ,
1200 END  ! End of Task 1
```

7.10 STRINGS

A String is a form of data made up of alpha-numeric characters. Strings can be regarded as either constants or variables in the same way as numeric data. Upper and lower case letter may be used in defining a string.

7.10.1 STRING CONSTANTS

String Constants are used where a specific message is to be printed or displayed. They are always enclosed in quotation marks which define start and stop points of the string. Only "Printable" ASCII characters can be used as String Constants.

An example of their use is as follows:

```
10 PRINT "THIS IS A STRING CONSTANT"
```

When this one line is run the string between quotation marks will be printed on the terminal screen. The DISPLAY command will present the string to the Engineers Panel.

7.10.2 STRING VARIABLES

Strings may be viewed as variables by the assignment of an alphabetic character, A through Z, followed by ($). The $ defines that the subsequent data within quotation marks is a string variable. The alphabetic character preceding the $ symbol define which variable. String variables can contain any ASCII character.

7.10.3 STRING DIMENSIONING

String Variables may have up to 50 elements in one dimension only. Each variable element may contain a maximum of 80 characters. However, the product of the number of elements and the number of ASCII characters must not exceed 2047. Initially, there is only memory storage space allocated for Numeric Variables. This must be modified using the DIM statement before using String Variables.

```
10 DIM A$(10),32 <── # of ASCII
          ^         characters in
          |         each String
# of Dimensioned   Variable
String Variables
```

Note: The product of elements and characters = 10 x 32 = 320. This is legal since 320 < 2047.

7.10.4 ASSIGNING STRING VARIABLES

String variables are simply assigned by using the equals statement. A string variable can be assigned to another string variable or to a CONSTANT string.

```
A$(2)  =  B$(30)
A$(3)  =  "HELLO"
```

7.10.4 ASSIGNIMG STRING VARIABLES CONTINUED

To distinguish alpha string constants from **Series 100** program language, quotation marks are used.

```
A$(1)  =  "SP1"
```

As with a numeric variable, omitting the element number from the string variable label, refers to element (0).

```
A$     =  A$(0)
B$     =  B$(0)
B$(1)  =  B$(1)
```

7.10.5 MANIPULATING STRING VARIABLES

7.10.5.1 STRING ADDITION

The use of string variables becomes apparent when building messages to be displayed. We could for example assign: A$ = "THE EQUIPMENT" and B$ = "IS FAULTY" and, C$ = "IS OK".

If we RUN a program with the following lines:

```
50 D$ = A$ + C$     60 PRINT D$
```

We would see on the terminal display the summation of the two strings represented by A$ and C$, for example:

```
THE EQUIPMENT IS OK.
```

If, however, we replaced this arrangement with the following:

```
50 D$ = A$ + B$
60 PRINT D$
```

The Terminal would display:

```
THE EQUIPMENT IS FAULTY
```

This is a efficient and controllable way of manipulating messages.

7.10.5.2 RELATIONAL OPERATORS

The IF-THEN instruction can also be used with string variables. However, only the EQ operator is available. As an example of its use, consider:

```
10 IF A$ EQ B$ THEN GOTO 100
20 IF A$ EQ "ALARM" THEN GOTO 200
30 IF "HELLO" EQ B$ THEN GOTO 300
40 IF A$ + B$ + "YES" EQ C$ THEN GOTO 400
```

Variables and constants can be on either side of Equation. Also, addition functions of variables and constants can be on the left hand side of Equation.

7.10.6 INDIRECT ADDRESSING OF VARIABLES

When ever a numeric or string variable is dimensioned as an array, a particular variable element in the array is addressed by the relevant element number. For example, consider the two dimensional numeric array:

```
A(0,0)     A(0.1)     A(0,2)     A(0.3)
A(1,0)     A(1,1)     A(1,2)*    A(1,3)
A(2,0)     A(2,1)     A(2,2)     A(2,3)
```

To load the variable location marked * with the number 23.5, the instruction,

```
10 A(1,2) = 23.5
```

could be used. The variable location, also, could have been loaded by:

```
10 C = 1
20 D = 2
30 A(C,D) = 23.5
```

The variable A(1,2) was indirectly addressed by variable A(C,D).

Indirect addressing is a very useful facility, particularly when manipulating variable arrays. To illustrate this, consider the requirement to load a 10 element array A0 - A9 with the same number. One method could be:

```
10 A(0) = 20
20 A(1) = 20
30 A(2) = 20
40 A(3) = 20
50 A(4) = 20
60 A(5) = 20
70 A(6) = 20
80 A(7) = 20
90 A(8) = 20
100 A(9) = 20
```

This takes 10 lines of program.

An alternative method is to use indirect addressing:

```
10 B = 0
20 A(B) = 20
30 B = B + 1
40 IF B LT 10 THEN GOTO 20
```

This only takes four lines.

Examples of the use of indirect addressing are:

```
10 A(1,2,G) = B(F)
20 IF A(G,C) GT 2.0 THEN GOTO 100
30 A(I,J) = C(I,J) * (D(I) + E(J))
```

7.10.6 INDIRECT ADDRESSING OF VARIABLES CONTINUED

The limitation of indirect addressing of variables is that the variable used as a pointer in the array, cannot be an array variable itself.

```
A (D(2)) = 2.0
A (G(I)) = 3.0
```

These are examples of illegal pointer variables.

Note: Remember, a variable name without an array reference equivalent to array location (0).

```
G = G(0) = G(0,0) = G(0,0,0)
```

When using indirect variable addressing two points to be careful of are:

1. The pointer variable should be an integer number. A(2.15) is an invalid element reference.

The Processor will automatically do an integer conversion on indirect array pointers.

```
10 B = 2.345
20 A(B) = 40
```

It will load variable A(2) with 40.

2. The pointer variable value should be in range of the array dimension. If it is out of range a DIM FAIL will occur. To illustrate, consider:

```
10 DIM A(10)
```

This creates variables A(0) to A(9),

```
20 G = 20
30 A(G) = 50
```

This, causes a DIM FAIL as element A(20) is out of the dimension range.

The above point is a common mistake made by programmers when using variable arrays. Very often, an array is created, and the array dimension is assumed incorrectly to be last element.

7.11 LADDER PROGRAMMING MODE

The **Series 100** System Ladder Diagram Package allows entry, editing and listing of Boolean Logic equations into a symbolic Ladder Logic format. Programming in Ladder Logic is done via a VDU terminal. This is feature is optional and nonstandard to the **Series 100.**

To select Ladder mode, type **LADDER(cr)** from the EDIT mode. Touch the ESC key to initialize the terminal. The Function Key descriptions will at the bottom of the screen. Also, note that the Baud Rate must be set to a minimum of 4800 for both the terminal and the processor. The Ladder mode may also be entered by setting the Boolean/Ladder Flag from the **UTILITY** heading of the Engineers Panel. To Return to the Boolean mode, type **BOOLEAN(cr)** to re-toggle the Boolean/Ladder Flag on the Engineers Panel.

7.11.1 PROGRAMMING WITH LADDER LOGIC

The VDU Function Keys represent various Ladder Elements. Most of these functions may be negated by holding the SHIFT key with a specific Function Key.

All lines in the Ladder Program must contain a line number. A line number may be initiated by pressing F1 START or SHIFT F1 START NOT. This will clear the screen and display a Ladder Contact with the cursor oscillating between the modes that the next OR Rung and the next AND Rung will start from. As more
elements are added, the cursor will continue to trackthe next OR/AND modes.

After entering a contact, a dash above it will indicate that its parameter must be entered. Any Input, Output, Flag, Timer or Counter Flag may be used.

F6 will CLOSE the bracket rungs. All rungs should be closed to the top rung level before the first TO (F7) or TO NOT (SHIFT F7) is used. Any rung detected by the Editor that is not closed at the first TO or TO NOT will automatically be closed.

NOTE: The ladder mode is optional and may be disabled the particular processor being used. This section may be skipped if desired.

Ladder Lines containing more than seven horizontal or vertical elements will cause screen corruption and incorrect print outs.

7.11.2 EDITING A LINE

Lines may be Edited by moving the cursor over the desired element or parameter via the four arrow keys.

Elements may be inverted or deleted. The parameter will be included and rungs will be closed automatically, as needed. Paramters may be overwritten or deleted via **DEL.** However, characters may not be inserted with **CNTL A.** When deleting, the element and its parameter wil be highlighted prior to confirmation. Entering **Y(cr)** will execute a **DEL.** Touching **(cr)** again will recast the altered rung on the screen with its new display.

7.11.3 PRINTING A LADDER PROGRAM

To print out a Ladder program, a printer must be attached to a terminal's auxillary port. Be sure that proper communications information is set before printing.

Note that lines originally written in Boolean form will have logic lines printed in Ladder form.

The PRINT command in the EDIT mode is the same as the LIST command. PRINT(cr) will print out the entire program. PRINT x,y will print a range of program lines. "x" is the starting line and "y" is the ending value. The Keyboard is locked out during PRINT to prevent accidental destruction of data.

7.11.4 SPECIAL KEYS

These are the special keys used for ladder programming. The Control Keys may be substituted for VDU's Function Keys.

Funtion Keys

F1	START Element	SHIFT F1	START NOT Element
F2	AND Element	SHIFT F2	AND NOT Element
F3	AND (Element	SHIFT F3	AND (NOT Element
F4	OR Element	SHIFT F4	OR NOT Element
F5	OR (Element	SHIFT F5	OR (NOT Element
F6	Close Bracket		
F7	TO Element	SHIFT F7	TO NOT Element
F8	INVERT Element (Only when editing a line)		

If F9 - F13 are present, these functions are available.

F9	LIST	SHIFT F9	RENUMBER
F10	DELETE Line	SHIFT F10	NEW
F11	LADDER	SHIFT F11	BOOLEAN
F12	PRINT		
F13	RUN		

CONTROL KEYS

CNTL B	Start Element
CNTL C	Start Not Element
CNTL D	AND Element
CNTL E	AND NOT Element
CNTL F	AND (note: This must be followed by CNTL B or CNTL C
CNTL N	OR Element
CNTL O	OR NOT Element
CNTL P	OR (note: This must be followed by CNTL B or CNTL C
CNTL Z	Close Rung
CNTL V	TO Element
CNTL W	TO NOT Element
CNTL Y	INVERT Element note: Only used when editing a line.
CNTL A	Alpha - numeric insertion mode note: Not available in Ladder Lines
CNTL Q	XON
CNTL S	XOFF

8.0 TROUBLESHOOTING

8.1 TROUBLESHOOTING

Series 100 system software has error detection capability. It displays faults on the Engineers Panel. Watchdog and Run LED's indicate operating conditions.

Here is a sequence that the user should follow is trouble with the **Series 100** occurs.

1) Check fuse located on Processor Front Panel for a total failure.

Failure of the Run LED to periodically flash indicates the user program is not running. This occurs when an error in new programs has caused the system to "lock-up".

2)Try to clear all error messages by pressing four buttons on Engineers Panel simultaneously. This will restart the system if failure is temporary.

Temporary error messages will usually be due to excessive electrical noise on power or signal line. Filtering or relocating the system will help if positioned near noisy equipment, such as unsuppressed switching contacts. If repositioning does not help, a filter should be installed in the cable supplying power to Processor.

3) Check for errors in the application program.

Program Error Messages are displayed when illegal lines of a program are entered. To isolate area of program responsible for error conditions it is suggested 'STOP' statements be imbedded throughout program. When program is in RUN and encounters a STOP statement, various parameters can be checked for error. Each program module can be tested, errors and correction can be made.

Use Engineers Panel to check on the status of Inputs and Outputs that do not respond or to which program does not respond.

4) Occasionally, total failure of Inputs and Outputs to respond is due to a faulty ribbon cable or connections. Check the orientation of the ribbon cable connectors and that a shorting plug is installed in last Module in chain. An Ohmmeter can be used to determine if break is present in any of the lines.

8.1 TROUBLESHOOTING CONTINUED

5)If a faulty module can be identified and confirmed by substitution or separate testing, it should be returned to Tenor Controls Service Department for replacement. Follow this procedure when returning items for service.

a) Contact Service Department with an Order Number for Module you need.

b) You will be invoiced against your Order for Module(s) you have requested.

c) Return your faulty module to Tenor. You will be credited for value of returned module, less any charge which may apply for repairing it. This does not apply for units under warranty where failure is due to faulty workmanship or materials.

8.2 ERROR MESSAGES

This list provides definitions and causes of the possible Processor error messages displayed during editing.

Brackets error: Too many brackets have been entered. Usually, an odd number of brackets is found.

Buffer full: Processor's user input buffer being full.

Checksum error: Incorrect programming from HHP or memory exhausted in 1 Processor.

Command error: Processor unable to recognize the command.

Deleted: Yes confirmation caused some program deletion.

End of file: Marker for End of File missing when programming from HHP.

Error 3: Line compilation exceeds 255 Bytes. Incorrect line must be re-written, possibly to occupy two lines.

Error 12: Line compilation and "packaging" data exceeds 255 Bytes.
Same solutions as Error 3.

Invalid number: Entered number too large or too small for valid range.

Line cancelled: Confirmation of line being cancelled as a result of 'Control X' line cancel operation.

Line missing: Branch instruction attempts to restore program to a missing line.

8.2 ERROR MESSAGES CONTINUED

Line number: Execution of an infinite loop was found where GOTO or GOSUB instructions have equal source and destination line numbers.

Memory full: Available Memory limits exceeds; insufficient space to accommodate line which has been submitted to Editor.

Memory missing: Detection of EPROM instead of RAM During LOAD from HHP to Processor.

Module number: Range exceeded during function limit tests performed on entered Module Numbers, Flag Numbers and Counter Numbers. Occurs when any Module number greater than 31; Input/output number greater than 7; Flag number greater than 255; Counter/timer number greater than 63.

Output error: Boolean instruction contained no output statement.

Stopped at line: STOP instruction encountered as program is running.

Syntax error: Incorrect line composition.

Word error: Compilation of word is not recognized.

Yes or No: Processor seeking confirmation of some major programming request.

8.3 SYSTEM FAILURE

Application program execution will cease, all outputs including Watchdog are de-energized. An appropriate message is given. To clear, simultaneously press all four Engineers Panel buttons to recover from all failures. This will restart application program. If no programming device is connected, or enter the Editor if one is connected. Some failures may require program modification to remove the fault.

The possible failures are:

8.3 SYSTEM FAILURES CONTINUED

RAM FAIL - CMOS RAM checksum failure. Recovery by pushing all four buttons will set all I/O, Flag, Setpoints, Values and Variables to zero. Possibly due to discharged memory support battery. To recharge battery,leave system powered up for 3 to 8 hours. If faulty memory devices are found, replace memory or Processor as needed.

MEM FAIL - Program Memory checksum failure. Recovery by pushing all four buttons will clear Program Memory. Application program must be reloaded. Possibly due to discharged memory support battery or faulty memory devices.

I/O FAIL - I/O Module hardware integrity failure. May be caused by faulty I/O Module(s), ribbon connector(s), or lack of power to remote Input/Output module power supply. Check all Module to Node cable links and press all four buttons on Engineer's Panel.

W/D FAIL - Watchdog failure is caused by hardware or software timing error. It may also be caused by excessively long arithmetic statements. Long statements must be broken down into shorter statements on adjacent lines. If problem is not solved, faulty Processor should be returned to **Tenor Controls**.

N/A FAIL - Invalid command or address encountered. May be caused by wrong addresses in GOTO command. Check that a RUN has been executed on a programming device. Push all four buttons on Engineers Panel to recover.

VAR FAIL - Variable address out of allowed range. Program has referred to variable outside range of Variables dimension. To recover, modify or insert DIM statement for that Variable.

DIM FAIL - Variable dimension out of allowed range. Program has attempted to dimension variable beyond allowed range, or beyond memory capacity. Modify program to include proper range.

STK FAIL - Stack over/under flow error caused by unbalanced FOR...TO and NEXT commands, or existing FOR...TO loops without setting variable to greater than or equal to TO value. Adjust program accordingly.

OVF FAIL - Variable overflow. Application program should be prevented from taking variable beyond allowable values. Adjust program accordingly.

DIV FAIL - Illegal division by zero attempted. Check program for occurance of this condition and adjust accordingly.

8.3 SYSTEM FAILURE CONTINUED

INT FAIL - Internal hardware oscillator failure during Run.
Faulty Processor should be returned to Tenor.

OSC FAIL - Internal hardware oscillator fail at start up.
Faulty Processor should be returned to Tenor.

TASK N/A - Enabled task has not been defined in program. Adjust program accordingly.

NO TASKO - Task 0 missing from program. Insert Task 0 to program.

DUP TASK - Task number duplicated in program. Remove duplicate Task number from program.

DTA FAIL - Data exhausted in READ statement. Data must be restored if it is to be re-used.

FOR FAIL - Numeric values within Start Stop and Step beyond the range -32768 and +32768. Also occurs for Step values of 0 or incremented to overflow. Adjust program values accordingly.

COM FAIL - Main Processor cannot access second serial port to set up parameter information. Most likely caused by hardware failure. Verify port failure and return Processor as necessary.

TECHNICAL GLOSSARY

ALGORITHM - A sequence of instructions required to solve a problem.

BASIC - Beginners All-purpose Symbolic Instruction Code. BASIC is the language used to inform the Processor what to do.

CENTRAL PROCESSING UNIT (CPU) - The microprocessor/microcontroller that executes programs such as the Scheduler and the interrupt routines in the BASIC program.

IMAGE REGISTER - An area of memory used to store the current value of an external modules' inputs or outputs. The image registers are updated every time Task 0 reaches a break, but may change while Tasks 1 to 7 are executed. For this reason it is wise to place Boolean program lines in Task 0.

INTEGRATED CIRCUIT - An electronic circuit usually fabricated on a tiny piece of high purity silicon wafer.

INTERRUPT - A hardware means of interrupting the flow of a running program or Task, in order to do another. Most important interrupt is the 5 msec Timing Interrupt. Lowest priority interrupt is RS232 character receive and transmit routines.

MACHINE CODE - Language understood by the CPU.

MEMORY - Collection of integrated circuits capable of storing digital values with battery backup maintain their contents during a loss of power. This enables a power up in same state as it was when power failed or was removed.

MICROPROCESSOR - An integrated circuit used as the CPU.

MICROSECOND(µs) - 1.0 E-06 seconds, 0.000001 sec.

MILLISECOND(ms) - 1.0 E-03 seconds, 0.001 sec.

MULTI-TASKING - The Series 100 can have up to eight Tasks executing concurrently. The CPU can only do one thing at a time so the Scheduler passes each enabled Task in turn to the CPU to be executed. This happens so fast that the Tasks appear to be executing at the same time.

OVERHEAD - A word used to describe time used by the Processor that is not available for BASIC program execution, such as the time taken to do interrupt routines or external module updates.

PARSING - The process of breaking a BASIC program line into executable portions. Literally, to analyze a sentence in terms of grammar.

SCHEDULER - The software program inside the Processor which decides when and for how long each Task is executed, and interleaves this with the system overhead.

VARIABLES SPACE - Area of memory used to store current values of variables.

APPENDIX 1

ADVANCED SERIES 100 SOFTWARE FAILURES

1.1 THE VAR FAIL AND ITS CAUSES

During manipulation of variables, several error traps prevent accidental access to unintended locations. These cause the Processor to halt execution with a VAR FAIL. VAR FAIL will occur under the following conditions:

1 - When a variable is DIMENSION'ed it occupies locations in variable RAM space. If this space falls outside the allocated area, separately for numeric variables and string variables then a VAR FAIL occurs. This usually occurs when an attempt is made to allocate too many elements. There is space for approximately 1300 numeric variables when the data memory is configured for 8k and approximately 4000 when configured for 16k. Approximately 2000 ASCII characters can be stored in either configuration in string variables.

2 - When a numeric variable is accessed its location is checked to see that it falls within expected range; if it does not then a VAR FAIL occurs.

3 - When there is an attempt to do string manipulations on a numeric variable a VAR FAIL occurs.

4 - If the dimension of a numeric variable exceeds 84 then a VAR FAIL occurs.

5 - If an attempt is made to take the Square Root of a negative number then a VAR FAIL occurs.

6 - When manipulating Bytes, Words and Modules, an internal trap occurs with a VAR FAIL if the internal opcode becomes corrupted.

1.1.1 CLEARING A VARIABLE FAIL ERROR

To clear a VAR FAIL error press all 4 buttons simultaneously on the processor module.

If the VAR FAIL was caused by a faulty assignment of variable space the VAR FAIL can only be cleared by placing the processor module in the EDIT MODE and type "CLEAR".

1.2 THE STK FAIL AND ITS CAUSES

Internal software needs to temporarily store values and intermediate results in memory while it performs housekeeping functions of multiple, concurrent tasks. For this purpose a STACK is used. A stack is a data store organized as a First In Last Out queue; there are two of these main stacks.

THE TASK STACK - This stack stores the current line number of a task when the scheduler switches to executing another task. It stores the return address of a sub-routine, intermediate results of Part Complete operations such as square roots, and parameters of for/next loops.

THE BRACKET STACK - Brackets are used in mathematical formulae to force the order of execution of the individual statements. It is necessary to store the intermediate result while calculating the term within the bracket. The bracket stack is also used in complex operations such as indirect variable accesses and square root calculations as a temporary store.

1.2.1 The STACK FAIL

There are 32 bytes allocated to each of the stacks, 1 bracket stack and 8 task stacks, each having 32 bytes. Each operation requiring the stack checks whether there is enough room to store data in the stack, or, whether there is any data to take off the stack. Some illegal operations are picked up at Edit time such as nested brackets not matching, whereas others, such as too many nested GOSUB statements within nested for/to/next loops, can only be picked up at RUN time.

The Editor may tell you if the syntax of a line would cause a stack fail, whereas the RUN time errors will cause a STK FAIL to occur. In order to help calculate why a stack fail may occur, a table has been compiled showing the stack requirements of the appropriate statements. It should be noted that although there are 32 bytes available for each task stack, only 30 are actually usable by the task as 2 will be needed by the Scheduler when it switches tasks; also any part complete line uses 1 stack byte thereby reducing the usable number to 29 in this case.

SERIES 100 STACK UTILIZATION

INTRODUCTION	BRACKET STACK USAGE	TASK STACK USAGE
Numeric brackets of nesting	4 bytes per level	0
Indirect variable accesses, e.g. A = B (I)	4	0
Square root calculation	4 bytes used for temporary storage during calculation	8 bytes used while operation is Part Complete to enable other tasks to operate without being held up
Sub-routine calls	0	2 bytes used per level of call
For/Next loops to store loop start address,step value and limit value	0	6 bytes required
Dimension string or numeric variables	0	5
Input numeric	0	7
Input string	0	2
Print string constant	0	2
Print string variable	0	5
Print num variable	0	7
Async. comms.	0	3

1.3 PART COMPLETE LINES

This section details the types of Part Complete operations.

1.3.1 Long Line Part-Complete

This occurs where a line is made up of several instructions and parameters and the combined execution time of all the individual instructions is likely to exceed the time left after input/output updates before a 5msec interrupt occurs. An example line might be:

```
A = B + C * D / E
```

Individual arithmetic operations are carried out sequentially and between each, control returns to the Scheduler to carry out housekeeping. Long line part-completes can only occur in Tasks 1-7, they can only be interrupted by a Task 0 request, and the scheduler will return to complete the line before going on to any other task.

1.3.2 Instruction Part-Complete

An instruction such as Dimension takes a long time to execute and has been written to do so in small well defined chunks. During execution of this type of instruction, the scheduler retrieves parameters from the task stack, carries on with the next chunk and returns new parameters to the task stack for next time. This type of Part-Complete may be interrupted by other tasks.

1.3.3 Communications Part-Complete

For large interrogations/assignments via Serial Port B, the transfer of information to Serial B or assignment of variables takes place over several scans or in 10msec periods.

APPENDIX 2

CONTENTS

COMMUNICATIONS FORMAT

2.1 PIN CONNECTIONS

For RS 232 Communications, use these pin assignments:

```
Pin 1   GND  Ground, 0 Volts
Pin 2   TXD  Transmit Data
Pin 3   RXD  Receive Data
Pin 4   RTS  Request to Send
Pin 5   CTS  Clear to Send
Pin 7   GND  Ground, 0 Volts
Pin 10  RS422 Data +
Pin 11  RS422 Data -
Pin 13  + 8.0 Volts
Pin 14  Edit Enable
Pin 20  DTR  Data Terminal Ready
```

Pins 1 to 7 (and 20) - Standard RS232 connections, Pin 14 is a special connection for the Processor.

Pins 1 and 7 are the 0 volts - Ground connection, for data and control signal reference.

Pin 2 - Transmit Data, is the data output and should be connected to the programming devices Receive Data, data input connection.

Pin 3 - Receive Data, is the data input connection and should be connected to the programming devices Transmit Data, data output connection.

Pin 4 - Request to Send, this is a control output connection used to inform the receiving terminal that the Processor wishes to send data. This control output does not effect the operation of the Processor. It can be used with certain receiving terminals to tell them to get ready for data transmission. For example, it could tell a modem to dial and get a telephone line connection.

Pin 5 - Clear to Send, this is a control input to the Processor. If it is connected to the 0 volts, it will inhibit the Processor from sending data. If the terminal is left open circuit, a 4.7k ohm pull up resistor internal to the Processor will insure it is held at an enable transmission state. In certain circumstances it may be necessary to have this pin connected, for example, if Pin 4 is used to request a modem to dial up and connect a telephone line, then Pin 5 could be used to inform the Processor when connection is made.

Pin 20 - Data Terminal Ready, is a control output to inform a Programming device that the Processor is ready to receive data. As the Processor handles received data on a high priority interrupt, it is in most cases always ready to receive data. This output is always presented in a ready state.

Pin 14 - Edit Enable, is a special purpose control input for the Processor. If the input is connected to 0 volts, then the Processor will be able to enter its EDIT mode of operation. This is the mode in which it can be programmed by the programming device. The simplest way to enable editing is to short Pin 14 to Pin 1, Pin 1 being a 0 volt ground.

The simplest design of a RS232 interface cable, requires only three wires and a shorting link.

```
Processor                       Programing
Connection                      Device Connection

PIN 1 GND      0 DDDD?
PIN 14 EDIT    0 DDDDY
     ENABLE
PIN 3 RXD      0 DDDDDDDDDDDDDDDDDD 0 TXD
PIN 2 TXD      0 DDDDDDDDDDDDDDDDDD 0 RXD
PIN 7 GND      0 DDDDDDDDDDDDDDDDDD 0 GND
```

The Transmit Data output of Processsor is connected to Receive Data input of programming device. The Receive Data input of Processor being connected to Transmit Data Output. A common signal ground is achieved by connecting GNDs of both devices and finally in order to enable the Processor to enter EDIT mode, Pin 14 is shorted to Pin 1 on the Processosr 25 pin D type connector.

All control signal connections are left open circuit, which means control of data transfer is disconnected. In most cases of a simple VDU to Processor communication, this does not cause any problems. The VDU is always "ready" to receive data from the Processor. The Processor can easily process the manual input of keyboard entry.

For RS 422 communications, these are the significant lines:

```
PIN 14         TXD +ve
PIN 11         TXD -ve
PIN 16         RXD +ve
PIN 18         RXD -ve
```

In RS 485 applications, use:

```
PIN 10         DATA I/O +ve
PIN 9          DATA I/O -ve
PIN 17         EXTCLK +ve
PIN 15         EXTCLK -ve
```

ASYNCHRONOUS PROTOCOL: 7 or 8 bit ASCII with odd/even/mark/space parity bit, one/two stop bits at 110, 300, 600, 1200, 4800, 9600 Baud. Supports XON/XOFF and CTS/RTS handshakes .

SYNCHRONOUS DATA LINK: ISO 3309 uses 8 bit byte structure, except information field. A maximum of 128 bytes per frame are permitted. With self clocking, NRZI encoding mode is selectable. Multi-drops for RS 485 are supported to a maximum of 375kbps, self clocked or 500kbps, externally clocked.

4.2 SETTING UP A PROGRAMMING DEVICE

The first requirement is to set the data transmission format to 1 start bit, 7 data bits, 1 parity bit and 1 stop bit. This is normally achieved by selector switches or a 'set up' mode on the VDU (Programming Device).

Next select the required baud rate, the data transmission speed, between 300 and 4800 baud. The Processor is normally supplied with a baud rate of 1200 baud. If required this can be adjusted to match the programming devices baud rate via the Processor Engineers Panel.

Finally select the required parity check, ODD, EVEN or NONE, the **Series 100** is normally set to ODD parity at the factory. Again, if required it can be adjusted via the Processor Engineers Panel.

Mode Set Up Options

The ASYNCA Command allows the user to set up various parameters for communication of the **Series 100** processor. During power up, the port is configured to the default setting. The user may reconfigure if necessary. Keep in mind that changing these parameters may result is a loss of some data.

```
        ASYNCA p b s d /eo/
                       /so/
                       /xo/
```

Where:

p = o, e, m, s for Odd, Even, Mark or Space Byte Parity.
b = Baud Rate; 110, 300, 600, 1200, 2400, 4800, or 9600.
s = Stop Bits; 1 or 2 bits per byte.
d = Data Bits; 7 or 8 bits per byte.
eo = Echo Character Option; use lo for Line Echo Option.
so = Cyclic Redundancy Check
xo = Execute Option.

Echo Option returns an the character or line back to the sender when the successful communication is completed.

Cyclic Redundancy Check appends a transmitted message with a four charcter checksum. This checksum is the remainder formed by the preceeding data sequence by the CRC Polynomial, x E^16 + x E^12 + x E^5 + 1, to CCITT V41. Each four bit nibble of the CRC is sent as an ASCII representation of its hexidecial value. The CRC is generated from all characters generated from the preceeding message, including CR and LF's. **Error 1** will be returned on CRC characters do not match.

Execute Option requires messages to be acknowledged by a command when recived. A LF is returned upon successful transmission.

Response Times

A value is interrogated or assigned in one Scan. This is normally 10 msecond in length with Task 0. Single Bit Boolean variables are interrogated/assigned 8/scan; Multi Bit variables are handled +1/scan.

APPENDIX 3

MEMORY OPTION JUMPERS

3.0 MEMORY OPTION JUMPERSS

The **Series 100** processor modules are configured in the factory to contain the memory requirements requested by the user from the options listed in the sales literature. Some modification of the sizes and partitioning of the user program memory (PRAM) and in user variables memory (XRAM), are possible. Modifications involve fitting the correct memory devices, setting up the solder jumpers on the underside of the processor PCB and, selecting a wire jumper option on the processor board. Access to the processor PCB is achieved by removing the **Series 100** processor modules left hand plastic end cover. The PCB visible is the main processor PCB and should be carefully unscrewed and removed after disconnecting the three IDC connectors on the top edge of the board.

The following table indicates the memory required and, the underside solder PCB jumpers to be modified. On top, links 1 and 2 should be set to Link 2.

FIRMWARE ISSUE	RAM	XRAM	EPROM IC5	EPROM IC6	UNDERSIDE SOLDER JUMPERS 1	2	3	4	5	6	7	8	9	10
4.0	32k	8k	256	...	O	S	S	O	-	-	S	O	S	O
>4.1	32k	8k	256	128	O	S	S	O	S	O	S	O	S	O
>4.1	32k	8k	256	256	O	S	S	O	O	S	S	O	S	O
>4.1	24k	16k	256	128	S	O	O	S	S	O	O	S	O	S
>4.1	24k	16k	256	256	S	O	O	S	O	S	O	S	O	S

Symbols: > - Greater than or equal to

S - Shorted jumper position

O - Open jumper position

APPENDIX 4

ADVANCED SERIES 100 FIRMWARE INFORMATION

SERIES 100 EXECUTION TIMINGS AND THE W/D FAIL

LIST OF FIGURES

LIST OF TABLES

SERIES 100 EXECUTION TIMINGS AND THE W/D FAIL

The first section describes the operation of and interaction between the Scheduler and the Interrupt Routines, while the second section and appendices address the task of calculating BASIC program line execution times.

1.0 SCHEDULER AND INTERRUPT OPERATION

The Series 100 Scheduler along with a 5 msec Timing Interrupt routine controls the allotted Tasks of the CPU.

1.1 Power Supply Checks

Series 100 power supply has been designed to detect a power failure and enable the Scheduler to control the shut down of the system. Even when the supply has failed, there is enough decoupling within the power supply to allow time for the following sequence of events.

All outputs are turned off and the W/D output is de-activated.

Series 100 enters a safe state until power is restored.

When power has been restored, a checksum is calculated for the internal memory and checked against the stored data. If the checksum does not agree a RAM FAIL occurs.

1.2 INTEGRITY OF OPERATION

The CPU could crash in the event of mechanical damage or extreme electrical noise. The crash would be detected by the 5 msec Timing Interrupt and cause a W/D fail and regain control.

1.3 RS232 INTRRFACE RECEIVE AND TRANSMIT ROUTINES

These are handled by an interrupt routine of lower priority than the 5 msec Timing Interrupt. They occur as characters are received over the interface or when the transmit buffer can accept another character.

1.4 PROGRAM OPERATION AND SYSTEM MANAGEMENT

Tasks to be done are allocated time in the following manner. The program lines are not directly executable by the CPU and each operation or fragment must be translated into a call to machine code routines within the Processor's internal program memory. If any program line fragment takes longer than 5 msec to execute, the Timing Interrupt detects that the Scheduler has not checked the power supply. It also causes a W/D FAIL. The amount of time available to program lines can be calculated.

There are four types of program instruction as far as this discussion is concerned.

Normal - Operations, such as most maths functions, are executed line by line for each Task. They do not cause a return to the Scheduler to execute a different Task.

Break - A Break causes the Scheduler to set a new Task running, update the 63 internal timer counters, or idle until the next Timing Interrupt. The following instructions cause a break:

STOP	**NEXT**	**ENABLE TASK**
GOTO	**END OF TASK**	**PRINT**

Part Complete - Instructions that take too long to fit into the time available is called Part Complete. It must be done in several passes. They treat each pass as if it were a separate line, checking the power each time. Apart from INPUT, a Break is not caused by a Part Complete instruction. The following instructions are part complete:

SQR - Calculations, taking much time, cause this instruction to be Part Complete.

INPUT - The length of time for INPUT to execute is dependent upon external action. This instruction is the exception in Part Complete instructions. It also causes a break on each pass, allowing the Scheduler to execute other Tasks. This keeps Task with the Input instruction from degrading the performance of other Tasks.

DIMENSION - This instruction may take a long time to execute as it re-arranges the variable space to make room for the array being dimensioned.

BOOLEAN - The Boolean operators behave in the same way as normal instructions and do not cause a break. The Boolean operators are really machine code routines in themselves and are executed directly instead of interpreted. This leads to a very fast execution time.

2.0 SCHEDULER TIMINGS

Event 1 - This 5 msec interrupt signals the Scheduler it is time to execute Task 0. This interrupt, like every 5 msec interrupt, also updates the following signals to the Scheduler:

a) TASK 0 REQUEST - If Task 0 is not executing when an interrupt occurs, and the external module update has been done, this signal is set. The result is that Task 0 normally executes every 10 msec.

b) EXTERNAL MODULE UPDATE REQUEST - This is set by the Scheduler when Task 0 reaches a break. The next interrupt to occur will recognize this signal and update the external modules connected to the Processor from the image registers. It is this external module update that recognizes whether the cable connecting the modules has been damaged in any way. The external modules are connected in a ring network such that any break in the ring will cause the Processor to shutdown with an I/O FAIL.

c) 50 msec flag and 100 msec flag - These flags have the following effect upon the Scheduler:

If the 100 msec flag is set, it updates the 63 internal timer counters, and reads the real time clock.

If the 50 msec flag is set,it reads the Engineering Panel buttons and takes appropriate action.

If neither flag is set, then update the Engineering Panel display. This is the lowest priority of the three Tasks. These three Tasks are mutually exclusive, only one of them will be done on each loop round the Scheduler.

Event 2 - The Scheduler recognizes the TASK 0 REQUEST and executes Task 0 to a break. If Task 0 takes more than 5 msec to reach a break, then the Timing Interrupt will not have received an EXTERNAL MODULE UPDATE request and will only update the timing signals. If Task 0 takes more than 50 msec to reach a break, then the Engineering Panel display will not get updated if Task 0 takes more than 100 msec to reach a break, then the Enegineering Panel buttons will not be read and the internal timer/counters will slow down. The Task repeat timers for Tasks 1 to 7 will also slow down if Task 0 takes more than 10 msec to reach a break. As with any Task, the power is checked by the Scheduler at the end of each line of BASIC executed.

Event 3 - Task 0 reaches a break and returns to the Scheduler. The Scheduler then sets the EXTERNAL MODULE UPDATE request and examines the timing flags and takes appropriate action is done.

Event 4 - The Scheduler now executes Tasks 1 to 7 according to the following rules:

a) If a flag indicating which Task is interrupted is set, the partial result is retrieved and that Task is executed from than point in that line.

b) The power is checked by the Scheduler at the end of every line, regardless of whether it caused a Break.

c) If any Task repeat timer has timed out then that Task is executed to a Break or a Task 0 REQUEST.

d) If more than one Task has timed out, then they are executed in the order of highest priority.

e) A TASK 0 REQUEST will force a break from any of Tasks 1 to 7 at the end of a line fragment. Power is still checked here. If the break is forced in the middle of a line, the intermediate result is stored and a flag is set to retrieve the result on the next pass.

f) If no Tasks have timed out, then the Tasks are executed in the order of highest priority.

g) Every time a Task is executed, the repeat timer is reloaded with the value defined in the Task header.

h) If all Tasks 1 to 7 have been executed, the Scheduler starts again with Task 1 until a TASK 0 REQUEST has been received from the interrupt routine.

i) If Tasks 1 to 7 are not defined in the program, the Scheduler idles until a TASK 0 REQUEST has been received from the interrupt routine.

Event 5 - The interrupt routine recognizes the EXTERNAL MODULE UPDATE request and updates the image registers. This may occur during the timers/buttons/display routine or during Tasks 1 to 7, as it is synchronized with the 5 msec interrupt. The length of this action will depends upon how many external modules are connected. It is important to realize that if more modules there are present, then less time is available for each line of Basic to execute in Tasks 1 to 7. This is not true of Task 0 which always has almost 5 msec to execute each line. It is also important to remember that the routine can occur in the middle of a Boolean line in Task 1 to 7. In this case, the information in the image register at the beginning of the line may be changed by the end of the line. This cannot happen to a line in Task 0. The example in Figure A1 shows the external module update occurring under interrupt during the Tasks 1 to 7 period. If Task 0 had taken longer in time, the external module update would have occurred during the timers/buttons/display routine.

Event 6 - The interrupt routine finishes the external module update and updates the timing flags. The Scheduler is now returned to the timer/button/display routine or Tasks 1 to 7, whichever was operating when the external module update was done.

Event 7 - This interrupt recognizes that the external module update has been done and that the Task 0 request is not set. The interrupt therefore sets the TASK 0 REQUEST. This interrupt decrements the Task repeat timers and sets the timeout flag for each Task when it reaches zero. Therefore the Task timers decrement every 10 msec in a normal program, although this will get extended for a Task 0 that takes longer than 10 msec to reach a Break.

Event 8 - The Scheduler finishes executing the Task at the end of the line fragment where it detects the TASK 0 REQUEST. This is equivalent to event 2 in this sequence of events. The Scheduler continues this process with another Task 0 is detected or until a STOP is encountered or the escape key is pressed.

3.0 INSTRUCTION EXECUTION TIMES

This section explains how to calculate the execution time of a program line for a Processor.

A Boolean line has an overhead of 45 microseconds. Input functions take 5 microseconds, Output functions take 9 microseconds and brackets take 4 microseconds.

```
X01.0  AND  X01.1  OR  (Y02.1 AND NOT X01.2) TO Y02.0 TO NOT Y02.2
```

TABLE 1 BOOLEAN TIMING EXAMPLE

FUNCTION	INPUT	OUTPUT	BRACKET	TIME
X01.0	x			5 µsec.
AND X01.1	x			5 µsec.
OR (			x	4 µsec.
Y02.1	x			5 µsec.
AND NOT X01.2	x			5 µsec.
)				-
TO Y02.0		x		9 µsec.
TO NOT Y02.2	x			9 µsec.
Overhead				45 µsec.
Total				87 µsec.

The exceptions to this are the counters and timers, especially Counter 0. Output to Counter takes a maximum of 95 µseconds, Output to Counter/Timer Reset takes a maximum of 65 µseconds and Input from Counter 0 takes 15 µseconds.

3.2 BASIC LINES

A basic line has an overhead of 16 µseconds per instruction and 45 µseconds, overall. The length of each instruction is different and depends upon the type of variables involved. These lengths are given in the next section. Each line of basic is constructed in a similar way. When the Scheduler runs a program line, it runs a Line Parser. Parsing is the process of breaking a line into its constituent instructions and then executing them in the correct order. Here is a description of this in more detail.

3.2.1 PARSING A BASIC LINE

In Parsing an Aritmetic instruction, each instruction has a variable or constant associated with it. The variable after an = is related to an implied instruction, GET. The variable before an = is also related to an implied instruction, PUT. The sequence A = B is Parsed to become, GET variable B and PUT it into variable A. IF/THEN is only counted as one instruction and does not have a variable associated. Brackets are only counted as one instruction. Following are typical program lines and a table showing how they break down into instructions. Each instruction is a self-contained line fragment with an intermediate result held internally.

1. A = B + C
2. D(1) = E(1,2) * 3 - F
3. IF A GE B THEN GOTO 20
4. A = B * (C + D)
5. A$ = B$ + "ABC"

TABLE 2 BASIC EXAMPLES

EXAMPLE	INSTRUCTION	VARIABLE
EXAMPLE 1	=(PUT)	A
	(GET)	B
	+	C
EXAMPLE 2	=(PUT)	D(1)
	(GET)	E(1,2)
	*	3
	-	F
EXAMPLE 3	(GET)	A
	GE	B
	IF/THEN	
	GOTO 20	
EXAMPLE 4	=(PUT)	A
	(GET)	B
	*()	
	(GET)	C
	+	D
EXAMPLE 5	=(PUT)	A$
	(GET)	B$
	+	"ABC"
		(constant)

3.2.2 VARIABLE TYPES

Before proceeding with the next section it is important to understand the different types of variables available within the **Series 100** Basic language.

3.2.2.1 CONSTANTS

This is a number example, A = 1234, or B = 1.23e2 * -12.

3.2.2.2 SIMPLE VARIABLES

This is a variable denoted by a letter without an array element.

```
A = B + C
```

3.2.2.3 ARRAY

This is a variable with up to three array elements. These elements can be either direct or indirect. A direct element is an ordinary number. An indirect element is a simple variable, like A(1), B(2,3), C(4,5,6) are direct, A(Z), B(Y,X); C(W,V,U) are indirect and A(1,I), B(J,2), C(I,J,9) are a mixture. It is possible to combine indirect and direct elements in one array access. It will be seen in the next section that indirect variables are much slower to be accessed than direct variables. This is because an extra variable access must be done on j to get the element number.

4.0 PROGRAM MEMORY REQUIREMENTS

It is simple to work out memory requirements of a program. Each line breaks down into Operator/Parameter units and each operator takes a certain amount of memory. For a Basic numeric operator, it is dependent on the parameters; other operators are parameter independent. Each line or open bracket starts with an implied "GET", either Boolean or Basic.

```
30 X1.3 AND X1.5 OR (F3 AND NOT X1.2) TO Y2.0 TO NOT Y2.1
```

TABLE 3 EXAMPLE

OPERATOR/PARAMETER	LENGTH (bytes)
Overhead and X1.3	12
AND X1.5	5
OR (F3	17
AND NOT X1.2	5
)	-
TO Y2.0	6
TO NOT Y2.1	8
Total	53

TABLE 4

SERIES 100 PROGRAM AND MEMORY SPACE

Function	Length (bytes)
Program overhead	5
BOOLEAN	
Line overhead	12
Initial NOT	2
AND, OR, AND NOT, OR NOT	5
AND (...), OR (...)	17
AND NOT (...), OR NOT (...)	18
TO	6
TO NOT	8
ON, OFF	7
TO CU, TO CD	64
TO NOT CU, TO NOT CD	66
ON CU, OFF CU	65
TO NR	56
TO NOT NR	58
ON NR, OFF NR	57
TO NR0	4
TO NOT NR0	6
ON NR0, OFF NR0	
(Boolean) IF/THEN (Boolean)	13
(Boolean) IF/THEN (Basic)	12
(Basic) IF/THEN (Boolean)	5
BASIC	
Line overhead	5
Arithmetic Opererator Constant	4
Arithmetic Operator Variable, A	2
Arithmetic Operator Variable, A(I)	3
Arithmetic Operator Variable, A(I,J)	4
Arithmetic Operator Variable, A(I,J,K)	5
Arithm. Operator BI,BO,WI,WO,MI,MO	3
+, -, *, / (...)	2
Numeric functions as Arithmetic	
Operator String Constants	2 + length
Operator, String Variable, A$	2
Operator, String Varianle, A$(I)	3
Print/Display Numeric Var.	Arithmetic
Print/Display (;)	as Print/Display
GOTO, GOSUB	5
RETURN	1
FOR I/= X/TO Y/STEP Z 4 parts,	all presented as Arithmetic
NEXT I	same as Arithmetic
(Basic) IF/THEN (Basic)	1
INPUT Numeric Variable	1 more than Arithmetic
INPUT String Variable	1 more than String
TASK	3
ENABLE	2
END	1
STAT MICROSECONDS	2
AIN	3
AOT	2 + 4 Arithmetic
Dimension Numeric Variable	same as Arithmetic
Dimension String Variable	same as String
DATA	1 + 3/numeric and/or1 + length string
READ	+ Arithmetic and/or String
RESTORE	5

Arithmetic lines like, A = B + C) are coded as Implied instruction as, "GET" B + C = A).

Arithmetic operators are +, -, *, /, =, GT, GE, EQ, LE, LF, NE and the initial implied "GET". The comparison operators do not work with BI, BO, WI, WO, MI or MO.

5.0 GENERAL OBSERVATIONS AND PROGRAMING TIPS

The next section of this document shows that certain operations, such as division or indirect variable accesses, can take a considerable amount of time to execute. Some extreme combinations, such as division of indirect array variables by other indirect array variables that cannot fit into a 5 msec time slot and therefore cannot execute. If such an instruction is causing a Watchdog fail when it is in Task 1-7, then the problem might be cured by placing the instruction into Task 0. This is because no time is taken from Task 0 to do an external module update. When developing a program for the **Series 100**, remember that the more modules connected to the **Series 100**, the less time there is available for lines in Tasks 1-7. If possible, debug the program with the full complement of external modules connected, or with **Series 100** Simulator modules to match the final installation.

From Figure A1, it can be seen that an external module update will be done every time Task 0 reaches a break. The GOTO instruction causes a break, and so any Inputs could change after a GOTO. This can be used to minimize the temporary storage of inputs, since if we know that a break has not been executed then no inputs will have changed value. Alternatively, we can use the GOTO instruction to cause a break an thus force the execution of Tasks 1-7, the external module update, and the timers/buttons/display routine. A further feature of a Task 0 break, is that Task 0 cannot execute again until 10 msec later. This is often useful for inserting delays into routines to allow time,
for example, for relay contacts to close. The following lines will cause a 30 msec delay in Task 0:

```
10 GOTO 15 15 A=MI13.0
20 GOTO 25 25 B=MI13.0
30 GOTO 40
40 ! USEFUL PROGRAM GOES HERE
50 END
```

In the above program, the values of variables A and B could be different due to the break in line 20. However, if line 20 were deleted the values of A and B would always be the same, but the delay would be reduced to 20 msec.

6.0 INSTRUCTION TABLES

The information calculated is the absolute worst case for timing values. Tenor Controls can accept no responsibility for the use made of the information contained herein. This document is a resonable guideline and not a guarantee of performance. The following points should be to follow when developing programs for the **Series 100** to operate at top efficiency.

6.1 Keep the execution times of program lines as short as possible.

6.2 Split up complicated maths lines into several simple lines.

6.3 Use simple variables in maths lines.

6.4 Keep Task 0 down to 100 msec or less between Breaks, performance of internal counter/timers due to drift, and updates of the real time clock.

6.5 It should be reiterated that all of the times given below are absolute worst case. They have been calculated from the algorithms used within the **Series 100** internal software. However, many of these times will only occur in obscure cases, such as dividing an extremely large number by an extremely small number.

7.0 SAMPLE SCHEDULER TIMINGS

The line Update shows the external module updates and idle times. These idle times update the system timer and Task 0 request flag. Task 0 shows when it starts and finishes. It also sets the Update Enable Flag. Hskp shows the system housekeeping; the updating of the timers and counters, the reading of the engineering panel buttons and the writing of each of the Engineering Panel Display segments. Task n shows Tasks 1-7. 5 msec shows the 5 msec interrupt. T0, Reg and U ena are internal flags which tell the Scheduler what to do next. T0 Reg is set by an idle external module time and reset by the end of Task 0. U ena is set by the end of Task 0 and reset by an external module update.

Fig A1 - Diagram of Scheduler timings

Fig A2 - Pattern of housekeeping Tasks

Fig A3 - shows how a full complement of external modules cuts into time allowed for Tasks 1-7. Task or Scheduler space between Tasks is suspended until external module update has been completed.

Fig A4, A5 and A6 - show effect of extending Task 0 beyond 5 msec. To prevent a clash between external module update and Task 0, one of the external module updates is converted to an idle period. This slows down housekeeping. Timers are updated as close to 100 msec. as possible, as are Engineering Panel buttons (but with less priority). Display is not updated as quickly as before. As Task 0 gets longer, the amount of time spent on Tasks 1-7 shrinks. If Task 0 exceeds 50 msec. Engineering Panel will lock up and if it exceeds 100 msec. Timers will not be updated.

TABLE 5 INTERRUPT TIMINGS

Routine		Time	
RS232 Routine	Receive	78	µseconds
RS232 Routine	Transmit	64	µseconds

TABLE 6
EXTERNAL MODULE UPDATE TIMINGS

Routine	Time
Idle	201 µseconds
Update	42 per module + 246 µseconds
Update with Force	70 per module + 283 µseconds

TABLE 7 BOOLEAN TIMINGS

Routine	Time
(GET)	4 µseconds
(GET) NOT	6 µseconds
(GET) OP0	14 µseconds
(GET) NOT OP0	16 µseconds
AND	5 µseconds
AND NOT	5 µseconds
OR	5 µseconds
OR NOT	5 µseconds
AND OP0	15 µseconds
AND NOT OP0	15 µseconds
OR OP0	15 µseconds
OR NOT OP0	15 µseconds
TO	7 µseconds
TO NOT	9 µseconds
AND (...)	8 µseconds
AND NOT (...)	9 µseconds
OR (...)	8 µseconds
OR NOT (...)	9 µseconds
TO CU	89 µseconds
TO NOT CU	90 µseconds
TO CD	95 µseconds
TO NOT CD	96 µseconds
TO NR	65 µseconds
TO NOT NR	66 µseconds
TO NR0	63 µseconds
TO NOT NR0	64 µseconds

TABLE 8 ANALOG TIMINGS

Routine	Time Calculated	Measured
Analog in	72 µseconds	
Analog out		
(constant)	215 µseconds	336 µseconds
(variable)		
(simple)	382 µseconds	495 µseconds
(1 direct)	447 µseconds	546 µseconds
(1 indirect)	985 µseconds	936 µseconds
(2 direct)	482 µseconds	590 µseconds
(1 direct, 1 indirect)	1020 µseconds	980 µseconds
(2 indirect)	1558 µseconds	1371 µseconds
(3 direct)	544 µseconds	657 µseconds
(2 direct, 1 indirect)	1082 µseconds	1076 µseconds
(1 direct, 2 indirect)	1620 µseconds	1466 µseconds
(3 indirect)	2158 µseconds	1885 µseconds
(clock)	815 µseconds	662 µseconds
(sp, vu)	784 µseconds	361 µseconds
(vd)	819 µseconds	361 µseconds
(vu0)	831 µseconds	363 µseconds
(vd0)	859 µseconds	391 µseconds

TABLE 9 BRACKET TIMINGS

Routine	Time Calculated	Measured
Open brackets	98 µseconds	67 µseconds
Close brackets & ADD	653 µseconds	644 µseconds
Close brackets & SUBTRACT	686 µseconds	585 µseconds
Close brackets & MULTIPLY	1044 µseconds	553 µseconds
Close brackets & DIVIDE	3079 µseconds	1985 µseconds

TABLE 10 COMPARISON TIMINGS

Routine	Time Calculated	Measured
Compare constant	444 µseconds	440 µseconds
Compare variable		
(simple)	609 µseconds	607 µseconds
(1 direct)	674 µseconds	659 µseconds
(1 indirect)	1212 µseconds	1048 µseconds
(2 direct)	709 µseconds	701 µseconds
(1 direct, 1 indirect)	1247 µseconds	1092 µseconds
(2 indirect)	1785 µseconds	1482 µseconds
(3 direct)	771 µseconds	768 µseconds
(2 direct, 1 indirect)	1289 µseconds	1187 µseconds
(1 direct, 2 indirect)	1847 µseconds	1577 µseconds
(3 indirect)	2385 µseconds	1996 µseconds
(clock)	983 µseconds	639 µseconds
(sp,vu)	972 µseconds	507 µseconds
(vd)	1007 µseconds	588 µseconds
(vu0)	1019 µseconds	589 µseconds
(vd0)	1047 µseconds	551 µseconds

TABLE 11 DATA TIMINGS

Routine	Time Calculated		Measured	
Data	2	µseconds		
Read per entry				
(variable)				
(simple)	933	µseconds	528	µseconds
(1 direct)	998	µseconds	579	µseconds
(1 indirect)	1536	µseconds	970	µseconds
(2 direct)	1033	µseconds	623	µseconds
(1 direct, 1 indirect)	1571	µseconds	1013	µseconds
(2 indirect)	2109	µseconds	1404	µseconds
(3 direct)	1095	µseconds	687	µseconds
(2 direct, 1 indirect)	1633	µseconds	1075	µseconds
(1 direct, 2 indirect)	2171	µseconds	1496	µseconds
(3 indirect)	2709	µseconds	1887	µseconds
(t/c)				
(sp)	810	µseconds	617	µseconds
(vu)	898	µseconds	699	µseconds
(vd)	927	µseconds	728	µseconds
(vu0)	931	µseconds	727	µseconds
(vd0)	960	µseconds	756	µseconds
(string)				
(simple)	22*char+231	µseconds		
(array)	22*char+1465	µseconds		
Restore	58	microseconds		

TABLE 12 DIMENSION TIMINGS

Routine		Time	
Dimension variable	numeric	434	µseconds
Dimension variable	string	232+104 *No.of.elements	µseconds

TABLE 13 DIVISION TIMINGS

Routine	Time Calculated		Measured	
Divide by CONSTANT	3034	µseconds	1831	µseconds
Divide by VARIABLE				
(simple)	3201	µseconds	1986	µseconds
(1 direct)	3266	µseconds	2037	µseconds
(1 indirect)	3804	µseconds	2428	µseconds
(2 direct)	3301	µseconds	2117	µseconds
(1 direct, 1 indirect)	3839	µseconds	2536	µseconds
(2 indirect)	4377	µseconds	2918	µseconds
(3 direct)	3363	µseconds	2178	µseconds
(2 direct, 1 indirect)	3881	µseconds	2597	µseconds
(1 direct, 2 indirect)	4439	µseconds	2987	µseconds
(3 indirect)	4977	µseconds	3406	µseconds
(clock)	3595	µseconds	2115	µseconds
(sp or vu)	3564	µseconds	1975	µseconds
(vd)	3599	µseconds	1984	µseconds
(vu0)	3611	µseconds	2016	µseconds
(vd0)	3639	µseconds	2133	µseconds
Divide by BI/WI	3634	µseconds	2263	µseconds

TABLE 14 FOR-LOOP TIMINGS

A "FOR" statement has four parameters:
FOR (1) = (2) TO (3) STEP (4)
The overall worst case time is the sum of the overhead and the relevant 1, 2, 3 and 4 times.

```
10 FOR A = 1.000 TO B (1) STEP C(D)
 TIME  =  126  +  83 + 303 + 841 + 2095 = 3448  µsecond
```

Routine	Time			
	Calculated		**Measured**	
Overhead	2095	µseconds	1525	µseconds
1st				
(simple)	126	µseconds	143	µseconds
(1 direct)	191	µseconds	193	µseconds
(1 indirect)	729	µseconds	612	µseconds
(2 direct)	226	µseconds	237	µseconds
(1 direct, 1 indirect)	764	µseconds	655	µseconds
(2 indirect)	1302	µseconds	1074	µseconds
(3 direct)	288	µseconds	298	µseconds
(2 direct, 1 indirect)	826	µseconds	717	µseconds
(1 direct, 2 indirect)	1364	µseconds	1107	µseconds
(3 indirect)	1902	µseconds	1526	µseconds
2nd, 3rd or 4th				
(constant)	83	µseconds		
(simple)	238	µseconds	249	µseconds
(1 direct)	303	µseconds	299	µseconds
(1 indirect)	841	µseconds	718	µseconds
(2 direct)	338	µseconds	343	µseconds
(1 direct, 1 indirect)	876	µseconds	762	µseconds
(2 indirect)	1414	µseconds	1180	µseconds
(3 direct)	400	µseconds	398	µseconds
(2 direct, 1 indirect)	938	µseconds	817	µseconds
(1 direct, 2 indirect)	1476	µseconds	1208	µseconds
(3 direct)	2014	µseconds	1626	µseconds
(clock)	671	µseconds	259	µseconds

FOR-LOOP TIMINGS Continued

Routine		Time Calculated	
FOR	I = 1 TO 10 STEP 3	1917	µseconds
	I(2)	1967	µseconds
	I(A)	2386	µseconds
	I(2,3)	2011	µseconds
	I(A,3)	2429	µseconds
	I(A,B)	2848	µseconds
	I(2,3,4)	2072	µseconds
	I(2,B,4)	2491	µseconds
	I(A,3,C)	2881	µseconds
	I(A,B,C)	3300	µseconds
FOR I =	A TO 10 STEP 3	2083	µseconds
	A(2)	2133	µseconds
	A(B)	2552	µseconds
	A(2,3)	2177	µseconds
	A(D,3)	2596	µseconds
	A(D,E)	3014	µseconds
	A(2,3,4)	2232	µseconds
	A(2,E,4)	2651	µseconds
	A(D,3,F)	2042	µseconds
	A(D,E,F)	3460	µseconds
	TIME	2093	µseconds

TABLE 15 FUNCTION TIMINGS

Routine	Time Calculated		Measured	
ABS function				
(simple)	272	µseconds	273	µseconds
(1 direct)	342	µseconds	324	µseconds
(1 indirect)	934	µseconds	714	µseconds
(2 direct)	377	µseconds	367	µseconds
(1 direct, 1 indirect)	969	µseconds	786	µseconds
(2 indirect)	1558	µseconds	1205	µseconds
(3 direct)	436	µseconds	429	µseconds
(2 direct, 1 indirect)	1028	µseconds	847	µseconds
(1 direct, 2 indirect)	1620	µseconds	1238	µseconds
(3 indirect)	2212	µseconds	1657	µseconds
(clock)	824	µseconds	430	µseconds
INT function				
(simple)	1053	µseconds	481	µseconds
(1 direct)	1123	µseconds	532	µseconds
(1 indirect)	1715	µseconds	951	µseconds
(2 direct)	1158	µseconds	567	µseconds
(1 direct, 1 indirect)	1750	µseconds	994	µseconds
(2 indirect)	2294	µseconds	1413	µseconds
(3 direct)	1217	µseconds	637	µseconds
(2 direct, 1 indirect)	1809	µseconds	1056	µseconds
(1 direct, 2 indirect)	2401	µseconds	1446	µseconds
(3 indirect)	2993	µseconds	1865	µseconds
(clock)	1605	µseconds	523	µseconds
FRAC function				
(simple)	692	µseconds	505	µseconds
(1 direct)	762	µseconds	555	µseconds
(1 indirect)	1354	µseconds	973	µseconds
(2 direct)	797	µseconds	599	µseconds
(1 direct, 1 indirect)	1386	µseconds	1018	µseconds
(2 indirect)	1978	µseconds	1436	µseconds
(3 direct)	856	µseconds	660	µseconds
(2 direct, 1 indirect)	1011	µseconds	1079	µseconds
(1 direct, 2 indirect)	2040	µseconds	1469	µseconds
(3 indirect)	2632	µseconds	1888	µseconds
(clock)	1244	µseconds	553	µseconds
CHR$ function				
(simple)	660	µseconds	373	µseconds
(1 direct)	730	µseconds	424	µseconds
(1 indirect)	1322	µseconds	590	µseconds
(2 direct)	765	µseconds	468	µseconds
(1 direct, 1 indirect)	1357	µseconds	635	µseconds
(2 indirect)	1946	µseconds	914	µseconds
(3 direct)	824	µseconds	529	µseconds
(2 direct, 1 indirect)	1416	µseconds	696	µseconds
(1 direct, 2 indirect)	2008	µseconds	983	µseconds
(3 indirect)	2600	µseconds	1262	µseconds

TABLE 15 (CONTINUED)
FUNCTION TIMINGS

Routine	Time Calculated	Measured
SQR function First pass		
(simple)	2716 µseconds	
(1 direct)	2926 µseconds	
(1 indirect)	4765 µseconds	
(2 direct)	3031 µseconds	
(1 direct, 1 indirect)	4870 µseconds	
(2 indirect)	6700 µseconds	
(3 direct)	3208 µseconds	
(2 direct, 1 indirect)	5047 µseconds	
(1 direct, 2 indirect)	6886 µseconds	
(3 indirect)	8725 µseconds	
(clock)	3790 µseconds	
(sp,vu)	3712 µseconds	
(vd)	817 µseconds	
(vu0)	3868 µseconds	
(vd0)	3952 µseconds	
SQR function Part complete		
(simple)	5200 µseconds	
(1 direct)	5270 µseconds	
(1 indirect)	5883 µseconds	
(2 direct)	5305 µseconds	
(1 direct, 1 indirect)	5918 µseconds	
(2 indirect)	6528 µseconds	
(3 direct)	5364 µseconds	
(2 direct, 1 indirect)	5977 µseconds	
(1 direct, 2 indirect)	6590 µseconds	
(3 indirect)	7205 µseconds	
(clock)	5558 µseconds	
(sp,vu)	5532 µseconds	
(vd)	5567 µseconds	
(vu0)	5584 µseconds	
(vd0)	5612 µseconds	
SQR function Last pass		
(simple)	3592 µseconds	
(1 direct)	3732 µseconds	
(1 indirect)	4958 µseconds	
(2 direct)	3802 µseconds	
(1 direct, 1 indirect)	5028 µseconds	
(2 indirect)	6248 µseconds	
(3 direct)	3920 µseconds	
(2 direct, 1 indirect)	5146 µseconds	
(1 direct, 2 indirect)	6372 µseconds	
(3 indirect)	7598 µseconds	
(clock)	4308 µseconds	
(sp,vu)	4256 µseconds	
(vd)	4326 µseconds	
(vu0)	4360 µseconds	
(vd0)	4416 µseconds	

TABLE 16 GET TIMINGS

Routine	Time Calculated		Measured	
Get CONSTANT	77	µseconds	78	µseconds
Get VARIABLE				
(simple)	242	µseconds	244	µseconds
(1 direct)	307	µseconds	294	µseconds
(1 indirect)	845	µseconds	713	µseconds
(2 direct)	342	µseconds	339	µseconds
(1 direct, 1 indirect)	880	µseconds	757	µseconds
(2 indirect)	1418	µseconds	1176	µseconds
(3 direct)	404	µseconds	399	µseconds
(2 direct, 1 indirect)	942	µseconds	818	µseconds
(1 direct, 2 indirect)	1480	µseconds	1209	µseconds
(3 indirect)	2018	µseconds	1627	µseconds
(clock)	675	µseconds	427	µseconds
(sp,vu)	644	µseconds	373	µseconds
(vd)	679	µseconds	386	µseconds
(vu0)	687	µseconds	388	µseconds
(vd0)	715	µseconds	413	µseconds
Get MI	668	µseconds	618	µseconds
Get BI/WI	737	µseconds	618	µseconds

TABLE 17 GOTO TIMINGS

Routine	Time	
GOTO line number	23	µseconds
GOSUB line number	93	µseconds
RETURN from sub-routine	54	µseconds

TABLE 18 IF-THEN TIMINGS

Routine	Time	
If (basic)	9	µseconds
If (boolean)	14	µseconds

TABLE 19 INPUT TIMINGS

Routine	Time	
Input string		
(first pass)	446	µseconds
(middle passes)	395	µseconds
(last pass)	415	µseconds
Input variable		
(first pass)	446	µseconds
(middle passes)	395	µseconds
(last pass)	4415	µseconds

TABLE 20 SUBTRACTION TIMINGS

Routine	Time Calculated		Measured	
Subtract CONSTANT	622	µseconds	549	µseconds
Subtract VARIABLE				
(simple)	789	µseconds	716	µseconds
(1 direct)	854	µseconds	767	µseconds
(1 indirect)	1392	µseconds	1186	µseconds
(2 direct)	889	µseconds	811	µseconds
(1 direct, 1 indirect)	1427	µseconds	1230	µseconds
(2 indirect)	1965	µseconds	1648	µseconds
(3 direct)	951	µseconds	872	µseconds
(2 direct, 1 indirect)	1469	µseconds	1291	µseconds
(1 direct, 2 indirect)	2027	µseconds	1681	µseconds
(3 indirect)	2565	µseconds	2100	µseconds
(clock)	1183	µseconds	877	µseconds
(sp,vu)	1152	µseconds	894	µseconds
(vd)	1187	µseconds	881	µseconds
(vu0)	1199	µseconds	909	µseconds
(vd0)	1227	µseconds	948	µseconds
Subtract BI/WI	1220	µseconds	970	µseconds

TABLE 21 NEXT-LOOP TIMINGS

Routine	Time Calculated		Measured	
NEXT function				
(simple)	2504	µseconds	1935	µseconds
(1 direct)	2644	µseconds	2035	µseconds
(1 indirect)	3870	µseconds	2872	µseconds
(2 direct)	2714	µseconds	2123	µseconds
(1 direct, 1 indirect)	3940	µseconds	2960	µseconds
(2 indirect)	5160	µseconds	3797	µseconds
(3 direct)	2832	µseconds	2245	µseconds
(2 direct, 1 indirect)	4058	µseconds	3082	µseconds
(1 direct, 2 indirect)	5284	µseconds	3864	µseconds
(3 indirect)	6510	µseconds	4701	µseconds

Routine	Time	
Clear	115798	µseconds
Disable	4	µseconds
Stop	N/A	

TABLE 23 ADDITION TIMINGS

Routine	Time Calculated		Measured	
Add CONSTANT	621	µseconds	598	µseconds
Add VARIABLE				
(simple)	788	µseconds	765	µseconds
(1 direct)	853	µseconds	816	µseconds
(1 indirect)	1371	µseconds	1235	µseconds
(2 direct)	888	µseconds	861	µseconds
(1 direct, 1 indirect)	1426	µseconds	1279	µseconds
(2 indirect)	1964	µseconds	1698	µseconds
(3 direct)	950	µseconds	921	µseconds
(2 direct, 1 indirect)	1468	µseconds	1340	µseconds
(1 direct, 2 indirect)	2026	µseconds	1731	µseconds
(3 indirect)	2564	µseconds	2149	µseconds
(clock)	1182	µseconds	914	µseconds
(sp or vu)	1151	µseconds	882	µseconds
(vd)	1186	µseconds	894	µseconds
(vu0)	1198	µseconds	888	µseconds
(vd0)	1226	µseconds	851	µseconds
Add BI/WI	1219	µseconds	1017	µseconds

TABLE 24 PRINT TIMINGS

Routine	Time Calculated		Measured	
Print string constant	58*char+222	µseconds	57*char+224	µseconds
Print string variable				
(simple)	56*char+311	µseconds	55*char+302	µseconds
(array)	56*char+1545	µseconds	55*char+712	µseconds
Print accumulator				
(normal)	3816	µseconds		
(sci-format)	4025	µseconds		
Display string constant	17*char+169	µseconds		
Display string variable				
(simple)	15*char+258	µseconds		
(array)	15*char+1492	µseconds		
Display accumulator				
(normal)	3517	µseconds		
(scientific format)	3562	µseconds		

TABLE 25 PUT TIMINGS

Routine	Time Calculated		Measured	
Put to VARIABLE				
(simple)	775	µseconds	379	µseconds
(1 direct)	840	µseconds	379	µseconds
(1 indirect)	1378	µseconds	708	µseconds
(2 direct)	875	µseconds	471	µseconds
(1 direct, 1 indirect)	1413	µseconds	753	µseconds
(2 indirect)	1941	µseconds	1032	µseconds
(3 direct)	937	µseconds	534	µseconds
(2 direct, 1 indirect)	1475	µseconds	814	µseconds
(1 direct, 2 indirect)	2013	µseconds	1101	µseconds
(3 indirect)	2551	µseconds	1380	µseconds
(sp)	652	µseconds		
(vu)	740	µseconds		
(vd)	769	µseconds		
(vu0)	773	µseconds		
(vd0)	802	µseconds		
Put to Module out	457	µseconds	324	µseconds
Put to Byte out	830	µseconds	369	µseconds
Put to Word out	831	µseconds	367	µseconds

TABLE 26 STRING TIMINGS

Routine	Time	
Get constant	13*char+30	µseconds
Get variable		
(simple)	11*char+112	µseconds
(array)	11*char+1346	µseconds
Plus µseconds string constant	13*char+28	µseconds
Plus µseconds string variable		
(simple)	11*char+110	µseconds
(array)	11*char+1344	µseconds
Compare string constant	14*char+31	µseconds
Compare string variable		
(simple)	12*char+120	µseconds
(array)	12*char+1354	µseconds
Put to string variable		
(simple)	9*char+122	µseconds
(array)	9*char+1356	µseconds

TABLE 27 TASK TIMINGS

Routine	Time	
Task	23	µseconds
Enable	146	µseconds
End	6	µseconds
Stat	57	µseconds

TABLE 28 MULTIPLICATION TIMINGS

Routine	Time Calculated		Measured	
Multiply by CONSTANT	1013	µseconds	444	µseconds
Multiply by VARIABLE				
(simple)	1180	µseconds	611	µseconds
(1 direct)	1245	µseconds	662	µseconds
(1 indirect)	1783	µseconds	1081	µseconds
(2 direct)	1280	µseconds	706	µseconds
(1 direct, 1 indirect)	1818	µseconds	1125	µseconds
(2 indirect)	2356	µseconds	1543	µseconds
(3 direct)	1342	µseconds	767	µseconds
(2 direct, 1 indirect)	1860	µseconds	1186	µseconds
(1 direct, 2 indirect)	2418	µseconds	1576	µseconds
(3 indirect)	2956	µseconds	1995	µseconds
(clock)	1574	µseconds		
(sp or vu)	1543	µseconds		
(vd)	1578	µseconds		
(vu0)	1590	µseconds		
(vd0)	1618	µseconds		
Multiply by BI/WI	1611			

APPENDIX 5

A5.1 GENERAL TERMINAL REQUIREMENTS

Ladder programming requires an "adm31" terminal (Wyse-30 or Wyse-50) which recognizes the following escape sequences and cursor commands.

Cursor Control/ Escape Sequence	Action
Control K	Cursor up,
Control J	Cursor down.
Control H	Cursor left.
Control L	Cursor right.
ESC ?	Returns the cursor position in the form "rc<cr>" where re are the row and column of the cursor in HEX with an offset of 20H.
ESC = rc	Moves the cursor to the position given by rc
ESC M	Returns the character under the cursor.
ESC G n	Sets up an attribute (m) to run from the cursor to the next attribute or the end of screen.
ESC #	Locks the keyboard.
ESC "	Unlocks the keyboard.
ESC z text	Sets up the function keys and their messages.
ESC F text	Sends a message to the top of the screen.

Please read the following procedures and precautions before turning on the terminal.

1.If you have not already unpacked the terminal, carefully remove it from the container. Save all packing materials in case the terminal must be shipped again.

CAUTION: Sharp instruments should not be used to open the container.

Immediately notify the transfer company, if there is any damage.

2.Place the terminal on any sturdy table or desk.

3.Set the ON/OFF power switch on the front of the video module base to OFF by pushing the bottom of the switch.

4.Connect the keyboard cable to its receptacle on the video module base.

5.First connect the power cord to its receptacle on the video module base. Then plug it into a nearby three-pronged, grounded electrical outlet.

6.Connect the host computer communications cable to the modem port.

7.Connect the printer communications cable, if required, to the auxillary port.

After verifying that the terminal is properly installed, you are ready to proceed.

1.Turn on the terminal by pushing the top half of the ON/OFF switch.

2.Listen for an immediate beep. This indicates the power is on.

3.Watch for the cursor display in the upper lefthand corner of the screen.

If the CRT were warm, you would first see the screen flash several display patterns as the power-on self test is run.

4.Adjust the screen brightness with the intensity control on the front lower righthand side of the video module. Turn it downward for high contrast and upward for dim.

5. Swivel the video module left to right and tilt it up or down, until you find your personal comfort level.

The recommended height for the centre of the screen is 10 to 20 degrees below eye level. The keyboard whould be at or below elbow height.

6.To shut off the terminal, just push the bottom half of the ON/OFF switch.

The first time the terminal is turned on, a default set-up controls the way it operates for many variables called parameters. You can accept the default set-up or choose one to match your application program.

Whenever the parameters are changed, you can save the new choices so they will be in effect the next time the terminal is turned on; or you can easily return to the default set-up, if necessary.

1.Press SHIFT with SET-UP to display the configuration fields.

2.Press SPACE to display the next selection for a parameter.

3.Press CURSOR RIGHT to select the next field on the right.

4.Press CURSOR LEFT to select the next field on the left.

5.Press CURSOR DOWN to display the next level of fields.

6.Press CURSOR UP to display the previous level of fields.

7.Press SHIFT with SET-UP.

"Save changes for power-on?" flashes on and off.

8.Press Y or A to save changes in the set-up, or go to instruction 9.

A.If you press Y, all changes except those made to the function key definitions are saved for the next power-on.

B.If you press A, all changes are saved.

9.Press any other key instead of pressing A or Y to operate the terminal with the current parameter changes, but without saving them.

The next time the terminal is powered on the set-up returns to the configuration as it was before these changes were made.

10.Press ESC, with any of the levels of set-up parameters displayed, to call the default set-up.

HANDSHAKE=NONE SCREEN=80 CURSOR=BLOCK BLINK?=ON MODE=FDX

PARAMETER	SELECTIONS	EXPLANATION
HANDSHAKE	NONE (default)	Handshake protocol.
	XONXOFF	
	DTR	
	BOTH	
SCREEN	80 (default)	Screen column width.
	132	
	80 REV	
	132 REV	
CURSOR	BLOCK (default)	Cursor type.
	LINE	
BLINK?	ON (default)	Cursor display
	OFF	attribute.
MODE	FDX (default)	Communication mode.
	BLOCK	
	HDX	
	H-BLK	

DATA BIT=8 STOP BIT=1 PARITY BIT=NONE MODEM PORT BAUD RATE=9600

PARAMETER	SELECTIONS	EXPLANATION
DATA BITS	8 (default)	Character code length
	7	
STOP BITS	1 (default)	Character stop bits.
	2	
PARITY BIT	NONE (default)	Parity bit type.
	ODD	
	EVEN	
	MARK	
MODEM PORT	9600 (default)	MODEM port baud rate.
	19200	
	38400	
	50	
	75	
	110	
	134.5	
	150	
	300	
	600	
	1200	
	1800	
	2000	
	2400	
	4800	

```
BLK END=US/CR  AUTO NL=ON  CR=CR  AUTO SCRL=ON  AUX BAUD R=9600
```

PARAMETER	SELECTIONS	EXPLANATION
BLK END	US/CR	(default) End-of-block coding.
	CRLF/ETX	
AUTO NL	ON	(default) Automatic new line.
	OFF	
CR	CR	(default) RETURN action.
	CR,LF	
AUTO SCRL	ON	(default) Automatic scrolling.
	OFF	
AUX BAUD R	9600	(default) AUX port baud rate.
	19200	
	110	
	134.5	
	150	
	300	
	600	
	1200	
	1800	
	2000	
	2400	
	3600	
	4800	
	7200	

SCRL=JUMP STATUS=ON S.SAVER=OFF PROT=DIM TEST=OFF

PARAMETER	SELECTIONS	EXPLANATION
SCRL	JUMP	(default) Scrolling type.
	SM-1	
	SM-2	
	SM-4	
	SM-8	
STATUS	ON	(default) Message field display
	OFF	
S.SAVER	OFF	(default) Screen saver feature.
	ON	
PROT	DIM	(default) Protect attribute.
	REV	
	NORM	
TEST	OFF	(default) Diagnostic self test.
	ON (requires jumpers)	

KEYS?=US/UK RET/ENTER=CR/CR COMPATIBLE MODE=WY50 ENHANCE=OFF

PARAMETER	SELECTIONS	EXPLANATION
KEYS?	US/UK	(default) Language keyboard
	GERMAN	codes, (require
	FRENCH	special ROMs; US and
	SPANISH	UK are separate).
	DANISH	
RET/ENTER	CR/CR	(default) RETURN/ENTER action.
	CRLF/TAB	
COMPATIBLE MODE	WY50	(default) Compatible terminal
	TV1910	mode.
	TV1920	
	TV1925	
	ADDSVP	
	HZ1500	
ENHANCE	OFF	(default) WY-50 code enhance-
	ON	ment, (HZ-1500 and ADDS-VP)

The modem and auxillary port connector pin assignments are listed below. The interface cables must not have any wires running to Pins 9, 10, 11, 12, 14, 18, 19, 24 and 25 of the modem port.

	MODEM RS-232C		AUXILLARY RS-232C
Pin	Signal	Pin	Signal
1	Shield Ground	1	Shield Ground
2	Transmit Data		
3	Receive Data	3	Transmit Data to Printer
4	Request to Send		
5	Clear to Send		
6	Data Set Ready		
7	Signal Ground	7	Signal Ground
8	Data Carrier Detect		
*9			
*10			
*11			
*12	Leave unconnected		
*14			
*18			
*19			
20	Data Terminal Ready	20	Printer Ready
*24	Leave unconnected		
*25			

*DO NOT USE. If connected, improper video display will result.

The recommended settings for using Ladder mode are:

```
HANDSHAKE   XONXOFF
SCREEN      80
CURSOR      BLOCK
BLINK OFF
MODE        FDX
DATA        BIT 7
STOP        BIT 1
PARITY      ODD
BAUD RATE   4800
BLK END     US/CR
AUTO NL     OFF
CR          CR
AUTO SCRL   ON
AUX BAUD    300
SCRL        8M-8
STATUS      ON
SSAVER      ON
PROT        DIM
TEST        OFF
REUS        US/UK
RET/ENTER   CR/CR
MODE        WYSE50
ENHANCE     ON
```