1 MODEL 400 PROCESSOR OVERVIEW

1.1 Description

This bulletin describes the hardware and firmware features and installation procedures for the four types of SY/MAX® Model 400 Processors.

All versions of the SY/MAX Model 400 are single-width modules that may be installed into any SY/MAX digital or register rack assembly.

The Model 400 contains a versatile and powerful instruction set that performs over 100 functions. The memory in the Model 400 provides up to 16,000 words of user programming memory and allows control of up to 4,000 digital inputs and outputs in small to medium sized control applications.

Two serial communication ports on the Model 400 support a direct interface to other SY/MAX equipment to include programmable controllers and Network Interface modules (NIMs). The NIMs provide the SY/MAX devices with access to the SY/NET® high-speed network for local area network communication (LAN) facilities.

The Model 400 offers a real-time clock/calendar that may be used for interval timing of task execution and scanning operations. The clock/calendar and processor memory are additionally supported by an onboard lithium battery that allows the clock accuracy and contents of memory to be maintained when power is removed from the Model 400. The charge levels for the onboard lithium battery and the batteries in the SY/MAX power supply are monitored, and a front panel "BATTERY LOW" LED indicates when either or both batteries require replacement.

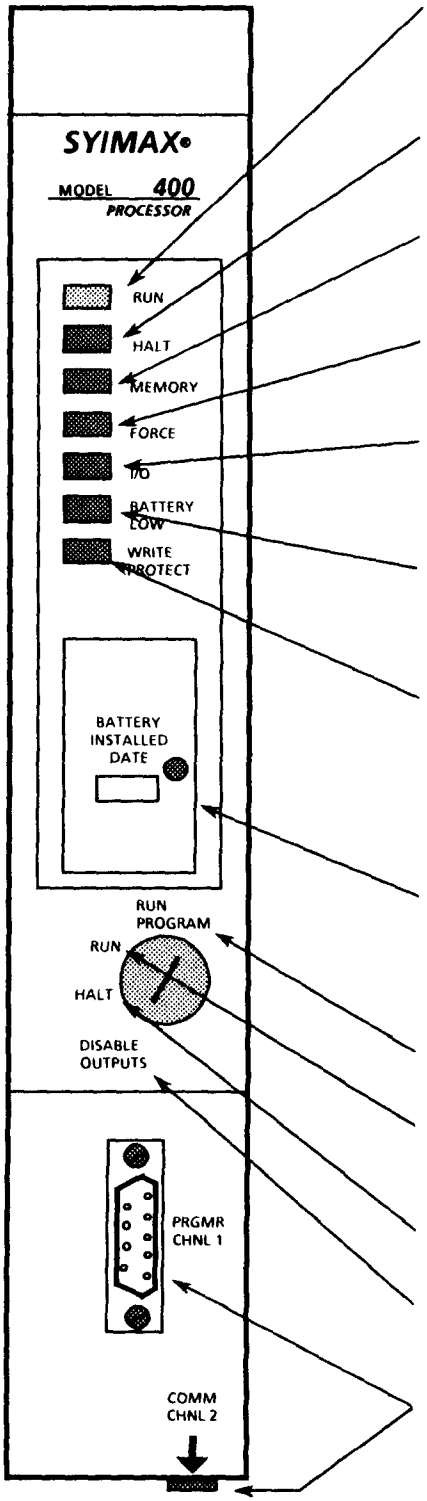
For added security, a "RUN" mode on the front panel keyswitch or an onboard jumper prevents online programming of the Model 400. A front-panel "WRITE PROTECT" LED indicates when the memory is write protected and whether the key switch or jumper is enabled. Software security (including program view inhibit) is also available through the use of a password.

CAUTION

The Model 400 is compatible with the family of SY/MAX devices and is physically interchangeable. If a Model 400 is substituted for another processor, the user must be aware that system response characteristics may change. Refer to Section 3.1 of this bulletin for a discussion of compatibility.

1.2 Front Panel Features

STATUS INDICATORS (LEDs)



RUN (GREEN) - When ON steady, the processor is scanning the ladder diagram program and operating on I/O's. When FLASHING, the processor is operating on ladder diagram program, but is not energizing any outputs (Disable Outputs mode).

HALT (RED) - When ON steady or FLASHING, the processor has halted its execution and is no longer scanning the ladder diagram program.

MEMORY (RED) - When ON steady, the HALT light also flashes, indicating the processor has halted due to a memory error. Refer to Appendix B, "Error Codes/Troubleshooting".

FORCE (RED) - When ON steady or FLASHING, one or several I/O have been forced to an ON or OFF state thereby overriding the ladder diagram program.

I/O (RED) - When ON steady the HALT light also flashes, indicating the processor has halted due to a malfunction in the I/O system. The I/O system is regarded as any I/O or register module that the processor controls in a rack assembly.

BATTERY LOW (RED) - When ON steady, indicates that the onboard lithium battery charge is low. When FLASHING, indicates that the power supply batteries are low. The ON steady condition takes precedence.

WRITE PROTECT (RED) - When ON steady, indicates that the user memory is protected by the keyswitch positioned in the RUN mode and/or the internal write-protect jumper is in the NO PROGRAM position. When FLASHING, indicates some type of software-generated security is in effect. The ON steady condition takes precedence.

BATTERY ACCESS DOOR - Used to gain access to the onboard lithium backup battery for installation and replacement.

KEYSWITCH

RUN/PROGRAM. In this mode, the processor executes the ladder program and allows changes to be made to the program.

RUN. *In this mode, the processor executes the ladder program but **does not** allow changes to be made to the program or forcing to be implemented or changed.*

HALT. In this mode, all outputs are turned OFF and the processor *stops* executing the ladder program.

DISABLE OUTPUTS. In this mode, all outputs are turned OFF and the processor continues to execute the ladder program.

COMMUNICATION PORTS

RS-422 serial differential port (COMM 1) for connecting programming and peripheral equipment. RS-422 serial differential port (COMM 2) for connecting inter-processor communications, printers, ASCII devices, D-Log modules, etc.

Figure 1.1 Front Panel Identification

2 SPECIFICATIONS

2.1 Model 400 Specifications

2.1.1 ELECTRICAL SPECIFICATIONS

Current Draw on SY/MAX Power Supply ..... 3500 mA (2500 mA for SCP-401). The PS-10, 11, 40, or 41 power supplies should NOT be used due to higher current requirements of the Model 400.

Memory Support Time From SY/MAX Power Supply Alkaline D-Cell Batteries .... 4 month minimum (battery age less than two years). Primary backup power source for ladder program, storage memory, and the clock/calendar when the Model 400 is in a rack during a power outage. Battery voltage can be monitored via contact status (bit 17 of reg. 8176).

Onboard Battery .. 3.5 to 3.6V AA lithium (Tadiran TL-5104 or SAFT LS6). Battery voltage can be monitored via contact status (bit 32 of reg. 8176).

Memory Support Time From Onboard Battery 1 year minimum (unloaded battery less than two years old). Secondary backup power source for ladder program, storage memory, and clock/calendar when the Model 400 is in a rack during a power outage. Primary power when Model 400 is removed from the rack. Battery is replaceable without powering down the Model 400.

Undervoltage Lockout Circuit .... Halts and resets the Model 400 and removes it from the SY/MAX bus when incoming DC supply voltage falls below 4.6VDC.

2.1.2 ENVIRONMENTAL SPECIFICATIONS

Ambient Temperature Operational ..... 32° to 140°F (0° to 60°C)

Storage ..... –40° to 176°F (–40° to 80°C)

Relative Humidity .. 5–95%, non-condensing

Vibration ..... Vibrated in three axes from 5 to 14 Hz at 0.2" (5 mm) displacement, and 14 to 200 Hz (two G acceleration maximum) for 90 minutes per X, Y and Z axis. Dwell at resonance frequencies for 10 minutes.

Electrical Noise ..... Passes the following tests for noise:

- NEMA Part ICS 2-230 and IEC 65A Part 5 Section 2.6.2.4. (Showering Arc).
- ANSI / IEEE C37.90a, IEEE 472 and IEC 65A Part 5 Section 2.6.2.5. (Surge Withstand Capability).
- Chattering Relay (Landis) Test.
- Current Spike (Reversing motor) test.
- ESD Protection meets IEC 65A Part 5, Section 2.6.2.2.

2.1.3 PHYSICAL SPECIFICATIONS

Dimensions
(without key) ..... Width- 1.5" (3.8 cm)
(Figure 2.1) Height- 12.8" (32.5cm)
Depth- 7.1" (18.0cm)
(with key) ..... Key adds an additional 1.0 inch (2.5 cm) to the depth when installed.

Weight (Approx.) .. 4.2 pounds (1.9 kg)

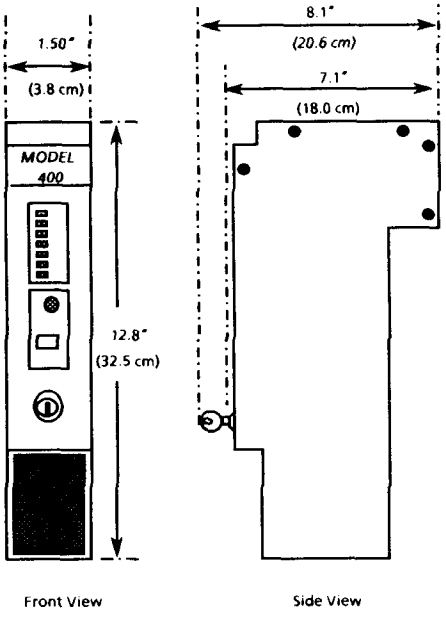


Figure 2.1 Model 400 Physical Dimensions

2.1.4 FUNCTIONAL SPECIFICATIONS

Scan Speed

SCP-401 ..... 4.1 milliseconds per K Boolean,
7.9 milliseconds per K nominal.

SCP-423, 424, 444 0.7 milliseconds per K Boolean,
2.9 milliseconds per K nominal.

NOTE: *I/O update from the image table for 128 local digital points will add a minimum of 0.3 milliseconds per scan. Refer to Section 14.2 for a detailed execution time description.*

Logic ..... Motorola® 68010 processor, Motorola 68881 math co-processor, and AMD 29116 for ladder scanning.

Real-Time Clock/Calendar .... Accuracy within one second per day @ 77°F (25°C), 16 seconds per day over full ranges:
temp. - 32 to 140°F (0–60°C)
relative humidity- 5–95%

Memory:

PROCESSOR TYPE	USER-MEMORY SIZE	MEMORY TYPE
SCP-401	4 K words	RAM
SCP-423	8 K words	RAM
SCP-424	16 K words	RAM
SCP-444	16 K words	RAM / UVPROM (up to 11K of memory can be stored in the UVPROM)

User Memory Utilization ..... 1 word per contact.

User Memory Overhead ..... 8 words

Front Panel LEDs .. RUN, HALT, MEMORY, FORCE, I/O, BATTERY LOW, WRITE PROTECT.

External I/O ..... Over 4,000 digital and over 3000 analog. Any mix in groups of 4, 8, 16, or 32 digital points or 4, 8, or 16 analog points per module.

Internal Relay Equivalents ..... 16 per unused register (4,000 nominal total).

Data Storage Registers ..... Up to 4000 16-bit integer registers or 2000 32-bit floating-point registers (floating-point not available in Type SCP-401).

Register Module Compatibility ..... All SY/MAX modules except Class 8030 Type CRM-115 and CRM-116 Bus Expander/ Terminator (see Section 3.1).

2.2 New Features

The following lists enhancements to the Model 400 that are not offered in the SY/MAX Models 300, 500, and 700 processors.

- **Real-time Clock** (Section 12.2).
- **Automatic Storage of Error Codes** (Control Register 8107).
- ***ASCII Input*** *(Section 10).*
- **Drum Sequencer** (Section 11).
- **Onboard Lithium Battery** (Section 12.1).
- ***RUN and RUN/PROGRAM Modes*** *(Keyswitch* Positions, Section 4).
- **Variable ROUTE Rungs** (Section 5.5).
- **Undervoltage Lockout Circuit** (Section 3.5).

2.3 Register Usage

The Model 400 addresses up to 4000 user-definable registers, each having a unique address value of 1 to 4000. These registers are located in an image table in the Model 400 processor and are used to reflect external I/O and internal relay status, store data, build shift registers, and accumulate time or count data in timers and counters. Additional registers are used to indicate overall programmable controller system status such as errors and keyswitch positions.

Each register is composed of 16 data bits and 16 status bits as illustrated in Figure 2.2. The data bits are accessible to the user for data storage and may be represented in binary or word form. The status bits cannot be changed or altered directly by the user, but can be programmed as contacts and monitored in the user memory.

Control registers 8097 to 8192 are used to control and monitor the basic operation of the Model 400. These registers are user-alterable, with exception of registers 8179 to 8192 which are read-only system status registers and can only be monitored. See Section 13 for details on the control registers.

The floating point (FLP) mode in the SCP-423, 424 and 444 processors combines two contiguous 16-bit data registers to form one 32-bit floating-point data register. FLP registers allow manipulation of numbers that contain floating decimal points. Refer to Section 7 for an explanation of "Floating Point Math" for the SCP-423, 424 and 444 processors.

Figure 2.3 illustrates the register usage for digital I/O points, analog I/O points, integer storage registers, and floating-point storage registers. Each digital I/O point occupies one bit of one register; 16 digital I/O points requires one complete register. Each analog I/O point occupies all 16 bits of one register and two complete registers are needed for each floating-point register.

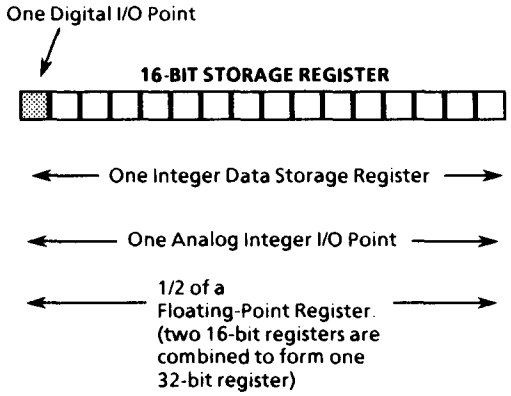


Figure 2.3 Register Use

The actual hardware I/O capacity of the Model 400 is over 4000 digital points and 3000 analog points. Typically, the maximum 16,000 word user memory is exceeded before the hardware maximum capacity is reached.

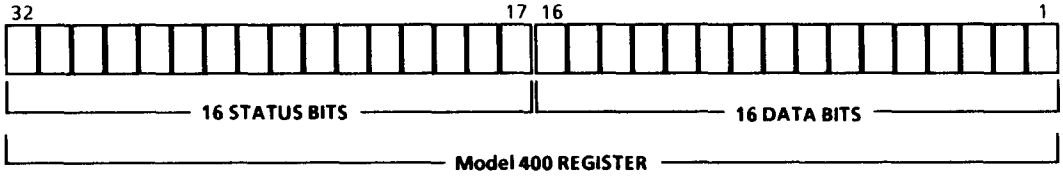


Figure 2.2 Model 400 Register Structure

2.4 Instruction Set for All Types

Figure 2.4 lists all of the instructions available in the Model 400 processors Types SCP- 401, 423, 424, and 444. The SCP-401 only handles integer data and the asterisk(s) following certain instructions (such as floating-point instructions), are not available in the SCP-401 and should be noted accordingly.

Figure 2.4 Model 400 Instruction Set

Instruction	Display	Description	Remarks
Contacts	⊣ ⊢ ⊣ / ⊢	Represents ON/OFF status of inputs, outputs, internal relays, and register bits. Normally OPEN (NO) or normally CLOSED (NC).	Input devices can be monitored in HALT.
Coil	—( )—	Represents an output connected to the processor.	Upon a transition to RUN, all coils are de-energized on the first scan. During the second scan, all coils are updated depending on the solution of the logic in that rung.
Internal Relays	R– —( )—	Represents control relay functions.	Operation is similar to an output, but output devices are not directly controlled.
Latch/Unlatch Relays	—(L)— —(U)—	Duplicates the action of a mechanically-held latching relay. This is accomplished through the use of two internal relay coils with the same address. Once a latch coil is energized, it remains ON until the unlatch coil is energized, even if power is removed.	It is recommended that latch and unlatch rungs be programmed *sequentially.*
Transitional Coil	—(T)—	An internal or external coil which is only energized on a transition of the logic from OPEN to CLOSED. The output remains ON for a single processor scan. The logic must turn the coil OFF and transition to ON again to re-energize the output coil.	Requires two consecutive I/O addresses, one to record the transition and one to remember the last state. A transitional coil *MUST* be assigned an ODD address number. The processor uses the next I/O address to record the last state of this same coil.
Master Control Relay	MCR —( )—	MCR ON allows logic to function. MCR OFF turns the outputs OFF. MCR controls all succeeding rungs until a new MCR coil is programmed or the end of the program is reached.	
Timer	TMR	Timer register stores a four digit decimal value from 0 to 9999. Selectable time bases are 0.01 sec., 0.1 sec., and 0.1 min.	Allows for programming ON/OFF delays and interval timers.
Counter	CTR	Counter register stores a four digit decimal value from 0 to 9999. UP, DOWN, and UP/DOWN counters are standard.	Counters can be cascaded to obtain a count greater than four digits.
Drum Sequencer	DRUM	Simulates the action of a rotary drum switch.	Manual or automatic time, event, or a combination of time/event sequencing.
Synchronous Shift Register	SHFT	Used to shift individual bits of a storage register in either a forward or reverse direction. 1, 8, or 16-bit channels can be shifted at a time.	Used in applications that require the user to keep track of bit patterns.
Asynchronous Shift Register (First In First Out)	FIFO	Data is loaded into the first zone and exits out the last zone using IN and OUT commands respectively. 8 or 16-bit channels can be shifted at a time.	Used to record the order of events as they occur, such as in conveyor or alarm annunciation.
Data Comparison	IF [=, ≠, ≥, <]	Compare functions between storage registers, between an integer and a constant, or between a floating-point storage register and a constant. When an IF comparison is true, the IF box acts like a short circuit.	Up to three comparisons can be programmed on one line. In *integer* mode, numbers from **–32,768** to **32,767** can be compared. In *floating point* mode, a range from -3.4×10^{38} to 3.4×10^{38} and values as small as $+/- 1.2 \times 10^{-38}$ are allowed. Math and SPECIAL functions can be used in IF statements.
Data Transfer	LET	Transfers data from one storage register to another or presets a constant into a storage register.	Ranges are the same as Data Comparison (above). Math and SPECIAL functions can be used in LET statements.

Figure 2.4 Model 400 Instruction Set

Instruction	Display	Description	Remarks
Transitional LET, IF	T LET T IF	Transition-sensitive; restricts the operation of the LET or IF instruction to one operation for each OPEN to CLOSED transition of the input condition.	
Indirect Register Read	RDSTAT	Indirect register read of *status* field.	
	RDDATA	Indirect register read of *data* field.	
	FNDBIT	Locate and clear the first bit (lowest bit number) set in the indirect register being read.	
Binary to BCD	BCD	Conversion of binary data in a storage register to its binary coded decimal (BCD) equivalent within a LET or IF instruction.	Allows the processor to drive a BCD-type LED display.
BCD to Binary	BIN	Conversion of binary coded decimal (BCD) data to its binary equivalent, within a LET or IF instruction.	Used in applications to convert the ON/OFF thumbwheel switch position status to binary form for processor readability and usage.
Standard Math Operations ***(Integer and floating point math*)***	LET, IF	Math operations are performed from left to right, and are programmed into LET or IF instructions.	The result of any *integer* math operation must not exceed **± 32,767**, or an overflow condition will result.
		Value range for *integer* math: ± 32,767	The *intermediate* result of an *integer* computation cannot fall outside **± 2,147,483,647**, or an overflow condition will result.
Addition	+	Value range for *floating-point* math operations: -3.4×10^{38} to 3.4×10^{-38}, not smaller than $+/- 1.2 \times 10^{-38}$.	The result of any *floating point* math operation must not exceed $+/-3.4 \times 10^{38}$ or be smaller than $+/- 1.2 \times 10^{-38}$, or an overflow condition will occur.
Subtraction	–	See Section 7, *"Floating-Point Math"*.	
Multiplication	X		The *intermediate* result of a *floating point* computation cannot fall outside the range of: $+/- 1.2 \times 10^{4932}$ to $+/- 3.4 \times 10^{-4932}$.
Division	÷		
Square Root	SQRT		
Absolute Value	ABS		
Change Sign	NEGATE		
Sine*	SIN		
Cosine*	COS		
LOG_{10}*	LOG		
Ln_e*	LN		
$(Y)^x$*	Y**X		

* *Not available in SCP-401*

Figure 2.4 Model 400 Instruction Set

Instruction	Display	Description	Remarks
Advanced Math and Trig Operations*	INVERT	Divide 1 by the current result.	* *None of these functions are available in the SCP-401*
	ROUND	Round floating point number to integer.	
	XSQRD	Square the current result.	
	HYPOT	Calculate 2-dimensional hypotenuse.	
	HYPXYZ	Calculate 3-dimensional hypotenuse.	
	TANGNT	Calculate tangent.	
	COSEC	Calculate cosecant.	
	SECANT	Calculate secant.	
	COTAN	Calculate cotangent.	
	ARCSIN	Calculate arc sine.	
	ARCCOS	Calculate arc cosine.	
	ARCTAN	Calculate arc tangent.	
	ARCCSC	Calculate arc cosecant.	
	ARCSEC	Calculate arc secant.	
	ARCCOT	Calculate arc cotangent.	
	SETDEG	Perform subsequent calculations in degree mode.	
	SETRAD	Perform subsequent calculations in radians mode.	
	DEGRAD	Convert degrees to radians.	
	RADDEG	Convert radians to degrees.	
	PI	Multiply result by pi (3.14159....).	
	L2T	Multiply result by log base 2 of 10.	
	L2E	Multiply result by log base 2 of natural log e.	
	LG2	Multiply result by log base 10 of 2.	
	LN2	Multiply result by log base e of 2.	
	EULER	Multiply result by Euler's Constant.	
	E	Multiply result by natural log e.	
	1DEG	Multiply result by 1 degree in radians.	
	1RAD	Multiply result by 1 radian in degrees.	
	ETOPI	Multiply result by e^{pi}.	
	PITOE	Multiply result by pi^{e}.	
	ETOE	Multiply result by e^{e}.	
	ETOEULR	Multiply result by e^{Euler}	
	SQRTE	Multiply result by square root of e.	
	SQRTPI	Multiply result by square root of pi.	
Statistical Operations*	ADDSTA	Store current result in statistical block of registers consisting of the summation of previous values, summation of the squares of previous values , and the accumulated count of previous values.	* *None of these functions are available in the SCP-401.*
	VARNCE	Calculate statistical variance on the accumulated totals in the statistical block.	
PID Computations*	PIDR	Calculate reverse-acting PID loop	* *None of these functions are available in the SCP-401*
	PIDD	Calculate direct-acting PID loop	
	PIDM	Calculate PID loop in manual	
	LEDLAG	Calculate lead/lag function for PID loop	

Figure 2.4 Model 400 Instruction Set

Instruction	Display	Description	Remarks
Accumulator Manipulations	YTOALT*	Same as (Y)^x except X in this case is taken from the alternate accumulator instead of the second argument.	All of these manipulations maintain an 80-bit precision when performing calculations that extend beyond a single rung.
	SWAP 16	Exchange low and high *bytes* of the current intermediate result.	* *These functions are not available in the SCP-401.*
	SWAP 32	Exchange low and high *words* of the current intermediate result.	
	ACCALT	Exchange current accumulator with the alternate accumulator.	
	ADDALT	Add the current intermediate result to the alternate accumulator	
	SUBALT	Subtract current intermediate results from the alternate accumulator.	
	DUPALT	Copy alternate accumulator into current accumulator.	
	DUPACC	Copy current accumulator (intermediate result) into alternate accumulator.	
	HYPALT*	Same as HYPOT except the alternate accumulator is used instead of the second argument.	
	ATANYX*	Similar to ARCTAN except that the function is performed on the ratio Y/X.	
Identify Number Type	NUMTYP	Identifies mathematical classification of the current intermediate result.	
Forcing	*CRT INITIATED*	Forcing overrides the actual input status or output as controlled by the ladder diagram.	Refer to Section 14.4.
Serial *Communication to* Other Processors or Peripherals		Transfer storage register and I/O status between devices which may be located up to 10,000 feet apart. All four are transition-sensitive.	Requires no additional interface hardware.
Read	T READ	Transfers storage register data from remote device to local processor.	
Write	T WRTE	Transfers storage register data from a local processor to a remote device.	
Alarm	T ALRM	Transfers a numerical value from a local processor to a remote device.	Numeric values from **–32,768** to **+32,767** can be transferred.
Print	T PRNT	Transfers ASCII-coded messages from the *processor to any device that communicates* in ASCII. See Section 13.2.	*Simplified means of producing alarm* messages, production reports, etc.
ASCII Input	T PRNT	Accommodates a block size of up to 256 characters with block termination based on either character count or a delimiter character. Supports various word structures including parity. See Section 10.	Allows the Model 400 to interface directly with bar code readers and weigh scales, etc.

Figure 2.4 Model 400 Instruction Set

Instruction	Display	Description	Remarks
AND	∧	A logical function used within an IF or LET command with the following property: if **X** and **Y** are two logic variables, then the function "X AND Y" is defined as: **X** **Y** **X AND Y** 0 0 0 0 1 0 1 0 0 1 1 1	Integer mode only.
OR	∨	A logical function used within an IF or LET command with the following property: if **P** and **R** are two logic variables, then the function "P OR R" is defined as: **P** **R** **P OR R** 0 0 0 0 1 1 1 0 1 1 1 1	Integer mode only.
Exclusive OR (XOR)	⊕	A logical function used within an IF or LET command with the following property: if **S** and **T** are two logic variables, then the function "S XOR T" is defined as: **S** **T** **S XOR T** 0 0 0 0 1 1 1 0 1 1 1 0	Integer mode only.
Random Number Generator	RANDOM	Generates a random integer number	Results in a positive random number that ranges from zero to the value of the argument minus 1.
GOTO	GOTO	Instruction to cause the processor to skip over a section of the program to another specified position in the program. Position is determined by a MARK rung placed in memory.	Shortens processor scan time during the immediate jump to a new rung position in the program. Can jump forward or backward.
Subroutines	GOSUB	Group of ladder diagram rungs that can be executed from the user's ladder diagram. Only enabled by a GOSUB instruction programmed in the main program.	Applications include variable sequence machines, recipe programs, and redundant processes.
	MARK ST SUB	Denotes the beginning of the subroutine area.	
	RTN	The end of a subroutine. Causes the processor to jump out of the subroutine and *return to the main program.*	

Figure 2.4 Model 400 Instruction Set

Instruction	Display	Description	Remarks
Timed Interrupt, Communication and Register Security, Scan Time Limit	S 8165	A special safeguard rung using register 8165 allowing the following features: 1. Communications Inhibit 2. Program Viewing Inhibit 3. Register Safeguarding 4. Timed-Interrupt Scan Control 5. *Scan Time Limit* (See Section 6.8 for details.)	Protecting programs. Used in conjunction with: **8176** **—( )—** **–16**
Matrix	M *LET M* IF M TIF M	A matrix is a group of storage registers handled as a unit. Formulates multi-register data transfers and math functions. Also allows multi-register compare statements.	Applications include machine diagnostics and menu selection, etc.
Array*	A LET A IF A	An array is a group of 32-bit floating point data storage registers operated on as a unit. Formulates multi-register data transfers, compares, and math functions.	* *Not available in the SCP-401*
Immediate I/O Update	S 8176 —( )— –07	Setting bit 7 of register 8176 to "1" (via a ladder rung) causes the Model 400 to interrupt its memory scan and immediately update selected external I/O points. Inputs are written to the image table and outputs are read *from* the image table.	Registers to be updated are user-defined.
Immediate *Communication* Update	S 8176 —( )— –11 or –12	Processes incoming data values from COMM *port channels which were received during* processor scanning.	

3 INSTALLATION

This section defines the following topics for the Model 400 processor:

- Differences in Operating Characteristics Compared to Other SY/MAX Devices and Certain Functions
- Rack Configurations
- Use of Register Modules
- Installation and Considerations
- Undervoltage Lockout Circuit Operation
- Power Installation Troubleshooting
- Rack Addressing the Model 400 Programmable Controller System

3.1 Differences In Operating Characteristics Compared to Other SY/MAX Devices and Certain Functions

The Model 400 is compatible with other SY/MAX devices, however, performance and/or functional characteristics in relationship to the SY/MAX 300, 500, and 700 processors may be different. Note that replacing another processor in an existing system with the Model 400 may require additional changes and the system may not operate exactly as before.

For example, the Model 400's user ladder program is executed based upon the state of an onboard "image table". The image table is updated at the end of every scan. This process is similar to the Model 300 operation but is different from the method used by Model 500 and 700 processors. The 500 and 700 use real-time bus updates; register information being used to execute the ladder program resides in modules outside of the processor and not in an internal image table.

In some cases, the programmable controller system may respond differently because of the Model 400's increased processor speed. Also, differences in how the Model 400 implements certain functions, such as forcing, produces results that are not the same as observed in a Model 500 or 700 system.

CAUTION

If the Model 400 is replacing a SY/MAX 100, 300, 500, or 700 processor in an existing system, READ the following SECTIONS to determine if certain processes must be changed before installing the Model 400.

The following modules produce characteristic differences when used with the Model 400:

1. **Type CRM-115/116 Bus Expander/Terminator Modules.** Due to characteristic bus timing delays, these modules are NOT compatible with the Model 400. In most cases, the Type EQ5138-G1/G2 Parallel Digital Driver/Receiver (PDD/PDR) modules can be substituted.

2. **Local Interface (LI) Modules.** These modules are compatible with the Model 400, but are NOT used to provide additional storage registers for the Model 400. The rack addressing that currently exists for another processor can be used with a Model 400 system. However, any registers that are assigned to an LI in the CPU rack and are *not assigned to remote drops* will adversely affect throughput.

3. **Local Transfer Interface (LTI) Modules.** These modules are compatible, however due to the image table in the Model 400, certain application considerations must be observed. Refer to Appendix C for operating considerations when using the Model 400 in a redundant LTI-equipped system.

4. **Parallel Digital Driver/Receiver (PDD/PDR) Modules.** These modules (Types EQ5138-G1/G2) are compatible with the Model 400 as long as certain rack addressing procedures are observed. Refer to the "Rack Addressing Register Allocation" discussion in Section 3.6.2.

NOTE: *If a Model 400 replaces a Model 500 or 700 in an existing system that contains PDD/PDR modules, rack addressing must be reviewed to determine if it is suitable or if alterations are required.*

The following functions produce different results when used with the Model 400 in comparison to other SY/MAX processors:

1. **Results of Forcing I/O.** Forcing in the Model 400 is implemented outside of the image table and is actually performed directly on the external I/O. It is important to note that the forced state may not be reflected in the image table. For example, the contacts of a forced output reflect the ladder logic state and not the forced state of the output. *Refer to Section 14.4 for more information.*

2. **Timed Interrupt Subroutine.** Due to the end-of-scan (EOS) update operation of the image table, a timed interrupt which executes more than once per ladder scan operates on unchanged external I/O information. The timed interrupt tolerance must now allow for the time required to complete an EOS update since an interrupt cannot be called while external I/O are being serviced. Refer to Section 6.8.1 for more information.

3. ***Matrix IF Instructions.*** *The Model 400 may not* scan some conditional elements in a rung that have no effect on the logical solution of the output. This optimizes scan time and increases processor scan performance.

   For example, if a rung contains a leading contact that is open, other conditional elements between the contact and the output element have no effect on its logical solution. If a rung contains a matrix IF instruction preceded by FALSE conditional logic, the matrix IF instruction is simply skipped over and not scanned. This differs from the Model 500 and Model 700, where bits 17 and 18 of the status register are reset when a matrix IF rung is executed FALSE. Thus, bits 17 and 18 in the status register of the matrix IF box should be ignored when using a Model 400 processor.

4. **Matrix Transitional IF Instruction.** As explained above, the Model 400 does not always scan all conditional elements. The matrix TIF instruction executes differently than for the Model 500 and 700. The matrix TIF instruction must be preceded by a transitional contact.

   NOTE: ***In order that the Model 400 behave as described in the programming manual, the Matrix TIF instruction must be preceded by a transitional contact (i.e., a contact that is energized by a transitional coil).***

5. **SCP-344 RAM/UVPROM Processor.** The SCP-344 requires the programming of two safeguard rungs that reference TLET S8164 and S8165. If a Model 400 replaces an SCP-344, these TLET rungs must either be removed or replaced with a TLET S8165 safeguard rung with new security features that are executed in accordance with Section 6.8.

6. **NUMTYP Instruction.** If the Model 400 replaces a Model 700 processor, the NUMTYP instruction yields slightly different results due to the difference in math coprocessors used. Refer to Section 8.6 for further details.

7. **Asynchronous Shift Register (FIFO).** If the Model 400 replaces a Model 300 processor, the FIFO operation is different in the handling of the last zone of the stack. In the Model 300, the DATA OUT zone is the last zone of the stack. In the Model 400 (as in the Model 500 and 700), the second-to-last zone is the bottom of the stack and the last zone is used as the DATA OUT zone. Because of this, the Model 400 are not accept a FIFO with a defined stack of less than three registers (ERROR 05 is generated). Refer to the programming Instruction Bulletin for more information.

8. **Memory Use.** If a Model 400 replaces any SY/MAX processor, more memory may be required to store the same program in the Model 400. For example, each new rung requires an additional 1/3rd of a word, while each new parallel branch now consumes a complete word. A good rule of thumb when transferring a program to a Model 400 is to allow 10% more memory than the size of the original program. See Section 14.1 for more details on memory usage.

9. **Counter Decode Equal to 0.** If the Model 400 replaces a Model 300 processor which has a program containing a counter rung with a decode of 0, the Model 400 de-energizes its associated output whenever the logic in the CLEAR line is false (and the count=0). Under the same conditions, the Model 300 energizes the output. Note that both processors energize the output when the decode value equals 0 and the logic in the CLEAR line is true.

10. **16 and 32 Function Digital Output Modules.** Revisions 1 and 2 of the Model 400 processor handle registers assigned to 4/8 and 16/32 function digital output modules differently **when processor scanning resumes after a CPU halt-to-run transition occurs**. The output register differences between the 4/8 and 16/32 function modules are dependent upon the type of rung that is used and the contact states of the rungs when processor scanning is halted, then resumed.

    If the outputs are programmed as regular coils, the 16/32 function digital output modules behave the same as 4/8 function digital output modules.

    When programmed as LATCH coils, UNLATCH coils, or when the digital outputs are driven directly by bits of a register (i.e., via LET statements), a difference between the 4/8 and 16/32 function output registers could occur for the following reasons:

    - The registers that are rack addressed for 16/32 function digital output modules are retentive, where the value in the register that existed at the time the processor halted, then resumed, is retained in the registers.

    - The 4/8 function digital output modules are reset to zero when processor scanning halts, then resumes and the value is not retained. Refer to Section 14.5, *Power Down Sequence*, for more information on the processor behavior during a processor halt-to-run transition.

11. **Editing in RUN/PROGRAM Mode.** If a Model 400 replaces a Model 300, 500, or 700 processor, the scan time may increase substantially for a single scan when an edit is performed while the processor is running. The difference in scan speed is a one-time occurrence and slower system response may be observed when editing the same program in a Model 300, 500, or 700 processor. Refer to Section 14.2.2 for additional information when editing in the RUN/ PROGRAM mode.

3.2 Rack Configuration

The Model 400 may be installed in the CPU slot of any SY/MAX register or digital rack assembly, with exception to the annunciator racks. The CPU slot is identified as slot R1 in each rack assembly (see Figure 3.1) and contains a rear edge connector that provides the appropriate bus lines, +5VDC, and ground for the Model 400 operation.

To directly control the various I/O modules configured in any programmable controller system, the Model 400 must be installed in the CPU slot or slot R1 of any digital or register rack assembly. Only one Model 400 may be installed in each rack assembly. Figure 3.1 illustrates the Model 400 installed in slot R1 of the various digital and register racks.

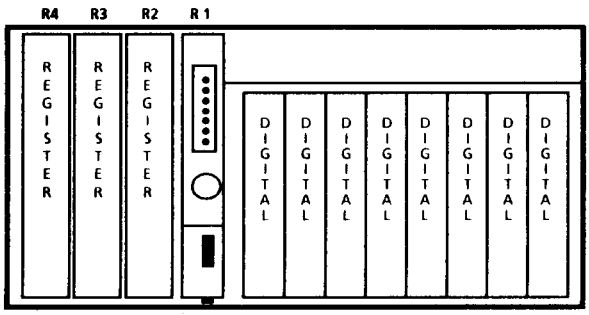
If the Model 400 is installed in a single rack configuration and only four-function and eight-function digital I/O are used, rack addressing is not required. The I/O addresses automatically revert to the default for the local I/O. If the Model 400 is installed in a multiple rack configuration and register modules (including 16 and 32-function digital I/O) are used, the corresponding registers and locations must be rack addressed. See Section 3.6 for details on rack addressing.

3.3 Register Modules

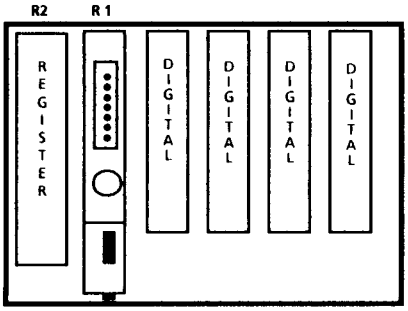
Register modules (also referred to as intelligent I/O) usually contain an onboard microprocessor and can handle more complex input and output data manipulation than the digital I/O modules. Register modules include analog I/O's, BCD multiplexed I/O's, and high-speed counter modules.

The register modules are installed in register slots in the various racks and depending upon the type of digital or register rack used, the number of register slots vary from 0 to 18. The following table and Figure 3.1 identifies the register slots in each type of digital or register rack:

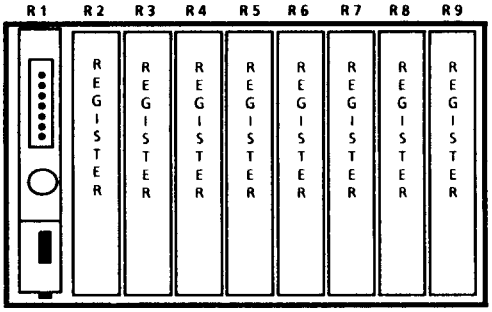
Class 8030 Type	# of Usable REGISTER SLOTS
CRK-100 (Digital Rack)	NONE
CRK-210, CRK-300, DRK-210, DRK 300, GRK-110, GRK-210, HRK-100, HRK-200 (Digital Racks)	1 SLOT (R2)
HRK-150 (Digital Rack)	3 SLOTS (R2-R4)
RRK-100, RRK-200 (Register Racks)	8 SLOTS (R2-R9)
RRK-300 (Register Rack)	15 SLOTS (R2-R16)



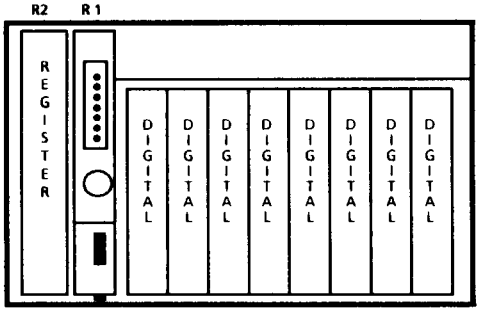
Eight-Function DIGITAL Rack (Type HRK-150)



Four-Function DIGITAL Rack (Type CRK, DRK, GRK)



REGISTER Rack (Type RRK)



Eight-Function DIGITAL Rack (Types HRK-100, 200)

Figure 3.1 Placement of Model 400 in a Digital or Register Rack Assembly

3.4 Model 400 Installation

The following procedure describes the Model 400's physical installation into a SY/MAX digital or register rack.

1. Remove the protective plastic film from the Model 400's faceplate.

2. Open the battery access door on the front of the Model 400 and observe the battery contact polarity as shown on the inside of the battery compartment cover. Install the lithium battery by matching the + and - polarities of the battery and the contacts in the battery compartment. Close the battery door and tighten the latching screw. Record the date of the battery installation on the space provided on the door.

3. If the Model 400 processor is installed in a CRK, DRK, or GRK four-function digital I/O rack, the *side plate shipped with the Model 400* (Part Number D30609-307-50) must be removed and replaced with an optional side plate P/N C30609-332-01A (contact Square D). The optional plate allows sufficient clearance between the side plate of the Model 400 and the edge connector in the rack.

4. The Model 400 *requires* +5VDC, *ground, and a* proper current rating to operate. Make sure that the SY/MAX power supply is capable of delivering the current required by all the devices in the rack.

   NOTE: *A PS-10, 11, 40, or 41 does NOT supply enough current for a Model 400 system. Refer to Instruction Bulletin 30598-159-xx to calculate the total current requirement for each module configured in the system and the appropriate power supply to use.*

CAUTION

Do NOT remove or install the Model 400 processor with power applied to the SY/MAX rack. Turn OFF the power at the power supply. Damage to the equipment may occur if the power is not removed from the rack before installation.

5. Make sure that the rack is mounted in an area where the temperature around the Model 400 never exceeds 140° F (60°C). Also, mount the rack so the Model 400 is installed in a vertical position.

WARNING

Do NOT remove or install a SY/MAX power supply or power cable to the rack with power applied. Turn OFF the power at the power supply and remove or unplug the incoming AC voltage line. Electrical shock, bodily injury, or damage to the equipment may occur if the power is not removed from the power supply before installation.

6. Use a Square D Class 8030 Type CC-10 power cable to connect the Model 400's rack to the SY/MAX power supply. Connect the CC-10 cable from the power connector on the rack to the P1 connector on the power supply. If the CC-10 cable is not used, the BATTERY LOW indicator on the Model 400 will not operate.

7. Locate slot R1 or the CPU slot in the rack assembly and install the Model 400 into the slot by applying steadily increasing pressure on the front of the processor. Press until the Model 400 is firmly seated in the edge connector and is against the stud located above the CPU slot.

   NOTE: *The Model 400 can ONLY be installed in slot R1 (the CPU slot) of any digital or register rack and only one Model 400 processor can be installed per each rack assembly.*

8. After the processor is properly seated in slot R1, close the latching bar located above the module slots (not applicable for four-function racks).

9. Tighten the captive screw at the bottom of the processor for a solid connection between the processor and the edge connector in slot R1.

10. Install the AC power line to the SY/MAX power supply and turn on the power switch on the SY/MAX power supply.

3.5 Undervoltage Lockout Circuit Operation and AC Fail Function

The Model 400 is designed with an onboard DC Undervoltage Lockout Circuit (ULC) that monitors the incoming DC voltage level at the edge connector in slot 1 of the rack. If the incoming DC voltage falls below 4.6 volts, the Model 400 enters a processor HALT state and disconnects from the bus in the CPU rack. If a DC undervoltage condition occurs, the processor is not warned ahead of time to execute an orderly shutdown.

When the incoming DC voltage rises above 4.6 volts, the Model 400 executes a standard power-up initialization sequence, but remains in the HALT mode. To resume the RUN state, the keyswitch must be toggled from RUN to HALT then back to RUN. When the ULC is enabled, an ERROR 902 is posted in register 8175 identifying the failure.

NOTE: *When the ULC system is enabled, none of the LEDs are illuminated on the Model 400.*

Since the ULC only resides in the Model 400, other devices located in the same rack may remain powered up. Modules under processor control behave as though the system is in HALT.

NOTE: *Non-bus modules, such as the D-LOG and Network Interface modules, may remain operating since their power requirements are less stringent than the Model 400's.*

AC FAIL FUNCTION:

The Model 400 also monitors an incoming DC signal off the edge connector of the rack identified as AC FAIL. AC FAIL is generated from all SY/MAX power supplies and indicates when the incoming AC voltage to the power supply has been lost or degraded.

The operation of the AC FAIL is different from the ULC where the AC FAIL signal alerts the processor to execute an orderly shutdown. The ULC does not alert the processor of an occurring DC power loss. Abnormal conditions such as an overloaded power supply or cable disconnection between the rack and the power supply can cause an interrupt of DC voltage.

As long as the Model 400 is receiving adequate power, at least one of the seven LEDs on the processor's front panel will be ON. If all of the Model 400 LEDs are OFF (or flashing erratically), a problem resides in the power source and should be repaired or replaced. Use the following procedures in Section 3.5.1 to determine the various power problems that may be the cause of a malfunctioning Model 400.

3.5.1 TROUBLESHOOTING POWER PROBLEMS

- If the system is powered up and none of the LEDs are illuminated on the Model 400, use the following procedure to determine the problem:

  1. Make sure the SY/MAX power supply is receiving proper incoming AC voltage

  2. Check the inline AC fuse and replace if blown. Before doing this, READ the WARNING below.

WARNING

Remove incoming AC power to the power supply before checking or replacing the AC inline fuse. Electrical shock or bodily injury could occur.

  3. Check that the power supply is delivering the proper amount of current for the system configuration. Refer to Instruction Bulletin 30598-156-xx or 30598-159-xx for a listing of current draws for various SY/MAX modules.

- If it is determined that the power supply is not the problem, use the following procedures:

  1. Check for +4.75 to 5.25VDC across pins 5 and 7 of the Model 400's serial port (COMM port) with a volt meter. Use Figure 3.2 for pin and signal identification.

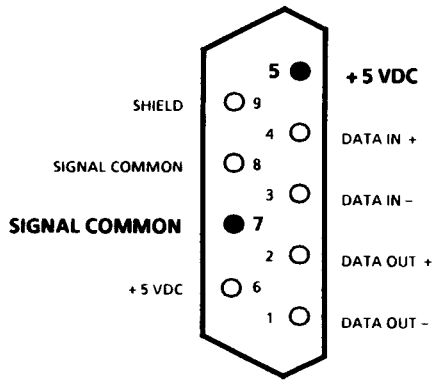


Figure 3.2 Serial Port Voltage Test Points

- If the voltage across pins 5 and 7 is zero, use the following to determine the problem:

Power is not reaching the rack:

1. Check that the CC-10 cable between the power supply and the rack is securely connected and is making good contact.

2. Visually check that all the pins in the connectors on both ends of the cable are the same height. If one or more are lower, pull the pin out until it is the same height as the other pins.

3. Check the continuity of the wires in the CC-10 cable with a meter. Replace the cable if an OPEN is detected.

4. Check the voltage at the P1 connector on the SY/MAX power supply. If voltage is NOT present, determine the cause and repair or replace the problem component.

DC power is being short-circuited:

1. Turn OFF power to the rack and check that all of the modules are properly seated. Turn ON power and see if the problem is resolved.

2. If step 2 did not resolve the problem, turn OFF power to the rack then remove and reinstall each module one at a time. Turn ON power to the rack after each module is reinstalled and check for proper voltage at each slot.

3. If a voltage problem is found in a slot of the rack or the rack has been damaged, replace the rack.

- If the voltage across pins 5 and 7 is greater than zero but less than 4.6 VDC, the ULC has properly prevented operation of the Model 400. Use the following procedure to identify why the low input voltage is being supplied from the power supply.

  1. As previously mentioned, the problem could reside in the integrity of the CC-10 cable or cable connections. Refer to steps 1 and 2 in the section **"Power is not reaching the rack"** to check the CC-10 cable.

  2. Check for excessive current draw from other modules in the rack. Check the current requirements for each module in the system and the power supply current rating (see Instruction Bulletin 30598-159-xx). If the power supply is rated under the total system current requirement, replace the power supply with an appropriately rated power supply.

  3. Once these possibilities have been eliminated, replace the power supply.

- If the previous troubleshooting procedures failed to resolve the problem, simplify the system by turning OFF the power to the rack and remove all of the modules from the CPU rack except for the Model 400. Turn ON the power to the system and see if the Model 400 powers up properly.

  Power OFF the system, reinstall the next module, and turn power ON checking for proper operation. Continue with this procedure for the modules that are to be reinstalled until the problem module is identified. Make sure that the power to the rack is turned OFF before removing or installing ANY module. This procedure offers the best systematic approach for isolating the power or component failure as quickly as possible.

3.6 Rack Addressing

A Model 400 Programmable Controller system consists of a Model 400 processor, digital and/or register racks with SY/MAX power supplies, and various interface and function modules. A great amount of flexibility exists when configuring and addressing Model 400 system components. Because of this flexibility, proper rack addressing is extremely important when installing the Model 400.

3.6.1 INITIAL SYSTEM LAYOUT

Rack addressing is determined by the physical layout of a programmable controller system, and is based on the following procedure.

1. Determine the physical location of the CPU rack and all other desired digital I/O and register racks.

   NOTE: *Remote racks can be located a total cable length of 10,000 feet (3050 meters) away from the CPU if LI/RI modules are used. PDD/PDR modules limit the distance between the racks to 6 feet.*

2. At each drop location, list the digital I/O requirements (type and quantity) along with any needed register modules. This determines the total number of modules, racks and power supplies needed.

3. Based on the total number of I/O modules, determine the number of Local Interface (LI) modules and Remote Interface (RI) modules needed. See Instruction Bulletin 30598-247-XX, "SY/MAX Local/Remote Interface" for further details.

3.6.2 PROCEDURE

The Model 400 executes the user's ladder logic program based on an image table. This image table, which can be visualized as being composed of external I/O registers, is updated (inputs are read from, and outputs are written to, the external world) at the end of every scan. An immediate I/O instruction can be user-programmed to momentarily interrupt a scan to update a user-defined portion of the image table (refer to Section 13.2 for information on bit 7 of register 8176).

The Model 400 has 4000 on-board registers; in this respect the Model 400 differs significantly from the SY/MAX Model 500 or 700. The Model 500 has 460 on-board registers but is able to address up to 2008; the Model 700, with no on-board registers, can address up to 8000. These "extra" registers used by Models 500 and 700 do not actually exist until other register modules such as a Local Interface are installed into the CPU rack.

The Model 400 is able to address all 4000 of its on-board registers. This means that it is no longer necessary to add Local Interface modules to obtain extra storage registers, since these extra registers already exist within the Model 400. Hence, the only purpose for adding a Local Interface module to the Model 400's CPU rack is for the serial communication link to any remote I/O.

Local Interface modules can be installed in slot R2 of the four-function and eight-function digital racks, and in any slot (except slot R1) of the HRK-150 and register racks.

NOTE: *In the RRK-300 rack, slots 17 and 18 cannot be used for a Local Interface.*

Sufficient registers should be assigned to each Local Interface module to account for all present and anticipated future external I/O, both digital and register, that will exist on remote drops.

NOTE: *A single Local Interface module can service no more than 255 external I/O registers.*

Refer to Section 14.2 for information on what effects rack addressing and register allocation have on system response, and Instruction Bulletin 30598-247-XX for more information on Local and Remote Interface modules.

3.6.3 DETERMINING RACK ADDRESSING NEEDS

Use the following procedure to determine addressing needs:

1. **Create a system layout sketch** showing the CPU rack, and all other racks with I/O wiring. This layout sketch insures proper assignment of racks to Local Interface channels. Each Local Interface module has two channels (1 and 2); each channel can address 8 racks (drops).

2. **Address the programmable controller system.** Install the Model 400 processor in the R1 CPU slot in either a digital or register rack. The rack addressing (RK ADDR) mode of the CRT programmer or SFW Programming Software can be used to allocate registers in the system as discussed below.

3. **Install other modules into the register slots of the CPU rack.** Addresses for the modules that are installed in the register slots should be assigned in ascending order. If the CPU rack is a digital rack, assign the CPU slot (R1) sufficient registers to handle the local digital I/O.

4. **Assign sufficient registers** to the Local Interface (LI) module to satisfy requirements for external remote I/O based on the content and quantity of remote drops.

5. **Assign registers to other register modules in the CPU rack** such as analog I/O's, BCD multiplexed I/O's and high speed counter modules in accordance with their Instruction Bulletins.

6. **Address remote digital I/O racks and remote register racks** within the range of registers previously assigned to the Local Interface module(s) during the CPU rack addressing. Local Interface modules communicate with these racks via a Remote Interface module located in each remote rack.

   Only those addresses assigned to the Local Interface slot in the CPU rack can be utilized by the remote racks. Assign registers to all Channel 1 drops sequentially, followed by all Channel 2 drops. There is no need to assign registers to the Local Interface modules to act as storage registers, since the Model 400 contains all needed storage registers.

   6A. Addressing the remote digital I/O racks. For all CRK, DRK, GRK, and HRK racks, slot R1 is always assigned as many registers as necessary to handle the local digital I/O. The Remote Interface module is always inserted into slot R1 of the remote digital I/O rack. Slot R2 is the register slot and should be addressed in accordance with the register module residing in that slot.

   6B. Addressing the remote register racks. Register rack slots are numbered from left to right; slot R1 is the leftmost. The Remote Interface module resides in slot R1; in the case of the register racks the Remote Interface is not addressed. Register modules are inserted into the slots to the right of the Remote Interface. The number of registers assigned to that slot is dependent on the individual register module requirement.

7. **Spare address registers,** for future expansion, can be assigned to any unused slot number for any remote drop. The slot that is assigned with the spare addresses does not even have to physically exist. For example, addresses can be assigned to slot R2 (register slot) of a CRK-100 (a rack without a register slot). These spare addresses act like internal I/O until they are used for I/O expansion and are reset each time the Model 400 is halted.

NOTE: *Any addresses assigned to remote drops utilize serial I/O update times as shown in the Local/Remote Interface Instruction Bulletin. These update times must be taken into consideration when calculating the throughput of a system.*

8. **Data storage registers** exist as a continuous "block" of registers. This block begins with the last address assigned to the CPU rack and ends with register 4000.

   NOTE: *Unlike the Model 500 and 700 processors, data storage registers do not have to be assigned to a slot to physically exist. All 4000 registers exist in the Model 400, they are ALWAYS available for use – even if not assigned via rack addressing. For example, if the last register assigned to a slot in the CPU rack is register 200, then registers 201 to 4000 are available for use as internal I/O or storage registers, despite the fact they haven't been assigned as such.*

3.6.4 RACK ADDRESSING REGISTER ALLOCATION

- *Default rack addressing (4000 registers in CPU slot R1)* applies when the Model 400 is installed in a digital I/O rack with no companion register modules and no remote I/O.

- If the Model 400 is installed in a digital I/O rack in a system containing register modules and/or remote I/O, assign enough registers to slot R1 for the local digital I/O.

- If the Model 400 is installed in a register rack, registers are NOT assigned to slot R1.

- Assign only necessary external I/O registers to Local Interface and other register modules in the CPU rack; include room for future expansion. There is no need to assign data storage registers to a Local Interface; this practice can adversely affect throughput.

- When assigning registers to a slot containing a Parallel Digital Driver module, some special application considerations are in order due to the image table operation of the Model 400. Observe the following rules:

    1. If expansion is to a single remote rack, the Parallel Digital Receiver MUST be connected to channel 1 of the Parallel Digital Driver. The CPU slot containing the driver should ONLY be assigned as many registers as required to accommodate the digital I/O in the remote rack.

    2. If expansion is to two remote racks, the rack connected to channel 1 of the Parallel Digital Driver MUST be an HRK-200 (the second rack can be any digital rack). The CPU slot containing the driver should be assigned as many registers as required to accommodate the digital I/O in the two racks. Thus, the first eight registers assigned will correspond to the I/O in the HRK-200 connected to channel 1, while the balance of the registers (up to a maximum of 16 if both remote racks are HRK-200s) will correspond to the I/O in the second rack.

NOTE: ***All registers assigned to the Parallel Digital Driver must have corresponding EXISTING external I/O. Failure to do this (for example, an HRK-100 rack connected to a channel receiving more than four registers) could result in oscillating outputs in the rack.***

Refer to Appendix D, Section D.2.3 for additional information on PDD/PDR updates.

- When assigning registers to digital I/O in a register module installed in the CPU rack, assign one register to a 16-point module or two registers to a 32-point module. Register modules should have registers assigned to the slot that they reside in.

- For remote **digital I/O racks**, registers for the digital I/O addresses should be assigned to the slot occupied by the Remote Interface module.

- For remote **register I/O racks**, do not assign registers to the Remote Interface. Instead, assign them directly to the slots in which the I/O modules are installed, including register modules that contain digital I/O.

- To improve throughput when laying out a Programmable Controller system, avoid "fragmenting" registers, i.e., ensure that all 16 bits associated with a register are either all inputs or all outputs. See Section 14.2.3.

- Any external I/O point can be forced ON or OFF. See Section 14.4.

- Model 400 system availability of 4000 registers is fixed; adding Local Interface or other register modules will not increase the available registers.

- Any of the 4000 on-board registers in the Model 400 that are not assigned to a slot are available as storage registers.

For more information on rack addressing, refer to Appendix D and the programming device's Instruction Bulletin.

3.6.5 RACK ADDRESSING COMPATIBILITY WITH OTHER SY/MAX PROCESSORS

The Model 400 can use programming and rack addressing configurations designed for other SY/MAX processors. An example of how this is accomplished is shown in Figure 3.3.

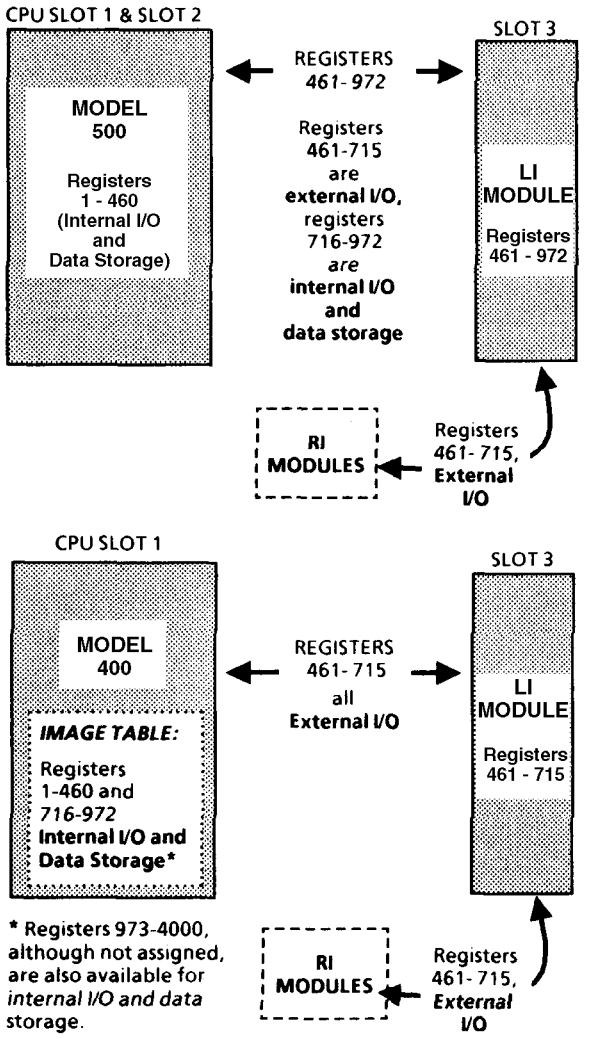


Figure 3.3 Model 400 vs. 500 Rack Addressing

Refer to Figure 3.3. A common Model 500 configuration is to assign the first 460 registers to slot R1, and registers 461 through 972 to a Local Interface in slot R3. *External* I/O registers in this configuration begin at register 461 and could extend up to register 715 (although consuming all 255 external registers, the maximum for a single Local Interface, would be unusual).

Internal I/O and storage registers in this example are located in the first 460 registers, and in registers 716 through 972. Note that for internal I/O and data storage, the Model 500 (or 700) will access registers that physically reside in the Local Interface module. The Model 400 uses its own on-board registers for this purpose, however. The only purpose for the Local Interface when used with a Model 400 is to provide communication between Remote Interfaces and the CPU rack.

Because of the image table, *only registers that are exchanged with a Remote Interface need to be updated by the Model 400 in the Local Interface.* It would be a waste of processing time and a large amount of unneeded bus activity to shuffle storage registers between the Model 400's image table and the Local Interface.

In the case of the "old" Model 500 addressing, the Model 400 is designed to accept the register allocation as defined, and will determine which registers *need* to be exchanged with the Local Interface. Since the Model 400 "knows" that the Local Interface cannot transfer more than 255 registers to the Remote Interface, it will transfer only registers 461 through 715 to the Local Interface.

This 255-register transfer occurs because the user has addressed 512 registers to the Local Interface. The same 255-register transfer would have occurred had the user addressed 2008 registers to the Local Interface (as could occur with a type CRM-211).

The important thing to keep in mind is that this Model 400 "decision-making" process is transparent to the user, and allows existing rack addressing to be used "as is".

When a programmable controller system is being newly configured for a Model 400, there is no need to follow old rack addressing conventions for assigning registers to Local Interface modules.

System throughput will be adversely affected if up to 255 registers are unnecessarily assigned to Local Interface modules located in the CPU rack. For best throughput, *only registers assigned to Remote Interfaces should be assigned to the Local Interface.*

The example Model 400 configuration pictured in Figure 3.3 shows the absolute maximum number of registers (255) being transferred between the processor and the Local Interface module. The throughput of this example could be improved substantially if only the registers associated with actual drops are transferred between the Model 400 and Local Interface, as shown in Figure 3.3A.

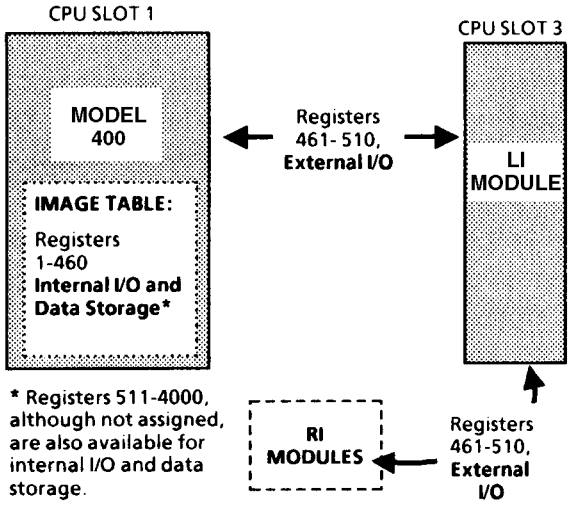


Figure 3.3A Model 400 Rack Addressed with 50 Remote Registers

If only 50 remote registers need be remotely addressed, the best configuration of the Model 400 shown in Figure 3.3 that would NOT require the re-addressing of registers in the program, appears as shown in Figure 3.3A. This is accomplished by assigning 460 registers to CPU slot 1, and 50 registers (461-510) to CPU slot 3. Registers above 510 would not need to be assigned, but would still behave as storage registers.

Refer to Appendix D, Section D.2.2 for additional information on the Local Interface module.

Rack addressing options when using existing programs in the Model 400 are therefore reduced to two:

1. **Preserve the existing rack addressing** for ease of transport between different processors, at a slight cost of throughput.

OR

2. **Modify the existing rack addressing** and register allocation to improve system performance.

CAUTION

If additional modules exist to the right of the Local Interface (LI) module being addressed by the Model 400, the addresses of these modules will change when the number of registers being assigned to the LI is changed. The Model 400's program must be altered to reflect the new addresses.

Refer to Appendix D, Section D.3 for a detailed discussion on the Model 400's compatibility with existing processor rack addressing configurations.

The system will operate properly with empty slots in the CPU rack, or the CPU rack can be altered to "close up" the empty slots. For example, since slot R2 is left empty when addressing a Model 500 or 700 system, the Local Interface could be placed in slot R2. An adjustment could then be made when rack addressing a Model 500-to-400 conversion to move all of the register assignments one slot to the left. This would free up an additional CPU rack slot, and eliminate the somewhat awkward condition of having slot R2 empty.

NOTE: *This procedure of moving the Local Interface to slot R2 would not require reallocating these registers – they would maintain their relative positions. It would, however, require adjustments to rack addressing and also that all modules in the CPU rack be moved one slot to the left.*

Refer to Appendix D for supplementary rack addressing information.

4 CONTROLS AND INDICATORS

4.1 Keyswitch Positions

The key-operated selector switch (Figure 4.1) locks the Model 400 into any one of four operating modes, each of which is described in this section. The key can be removed in any position.

IMPORTANT

Key position must be maintained for ½-second before that position will be recognized by the processor. Thus, ANY time the key passes through the HALT position, make sure the key remains in the HALT position for at least ½-second to allow the Model 400 to recognize the *transition.*

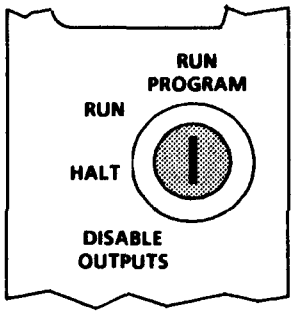


Figure 4.1 Model 400 Keyswitch

- **RUN/PROGRAM**

In this mode the processor functions the same as in RUN (below), but the program *can* be altered by the user via use of the override security position on the programmer. Forcing *can* be implemented and altered in this mode.

- **RUN**

In this mode the processor scans and executes the ladder program, and all external outputs are under control of the ladder program. While in this mode, the processor's program *cannot* be altered by any means. Forcing *cannot* be implemented in this mode; if already in effect, it cannot be altered.

IMPORTANT

When keyswitching between RUN and RUN/PROGRAM, do so as quickly as possible. A slow transition (greater than a quarter-second) may cause the Model 400 to go into HALT momentarily, then reinitialize before resuming RUN status.

- **HALT**

In this mode the processor is not scanning the program. All external outputs are turned off unless other special options are utilized.

- **DISABLE OUTPUTS**

In this mode the processor scans and executes the ladder program, but all external outputs are kept OFF. *Internal* outputs and storage registers (counters, timers, etc.) operate according to the program, as do the LEDs on I/O modules. This mode allows a program to be tested without actually energizing any external devices, thus eliminating the possibility of machine or process damage should the program be flawed in some way.

4.2 Indicator LEDs

As shown in Figure 4.2 on the following page, the Model 400 has seven LEDs located on the front panel. Each LED, except where noted, has three possible states: ON, OFF, or FLASHING. The following describes the various states of the LED indicators.

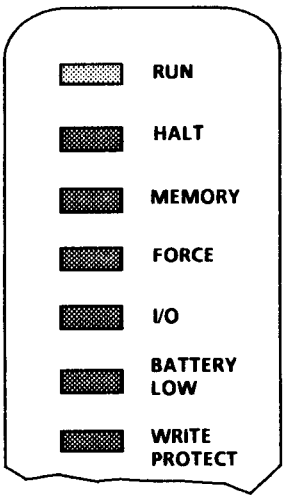


Figure 4.2 Model 400 Indicator Lights

IMPORTANT

If no LEDs illuminate or LED operation is erratic when power is applied to the Model 400, refer to Section 3.5.

As long as the Model 400 is receiving proper power from the SY/MAX power supply, *at least one* LED will be "ON" or FLASHING. If *all* LEDs (except RUN) come ON and stay ON, the Model 400 has failed to initialize properly. Check the processor-to-rack seating. An "all-ON" condition can also be caused by improper power supplied to the Model 400.

- **RUN**

ON: The processor is scanning the user program and is operating normally. This LED will light when the keyswitch is in either the RUN or RUN/PROGRAM position.

OFF: The RUN LED is off if the processor is halted by the keyswitch being in the HALT position, or if the processor is instructed to halt via the programming equipment or user program. This LED will also be OFF if the processor halts due to any Programmable Controller system malfunction.

FLASHING: The RUN LED flashes if the processor is in the DISABLE OUTPUTS mode due to one of the following conditions:

- Processor keyswitch is in the DISABLE OUTPUTS position.
- Processor has been instructed to run in the DISABLE OUTPUTS mode by the pro-gramming equipment.
- Processor has been instructed to run in the DISABLE OUTPUTS mode by the user program.

- **HALT**

ON: If the HALT LED is ON *continuously*, the Model 400's microprocessor is not operating.

Numerous diagnostic checks (power-up, halt-to-run, and run-time) are constantly made within the processor to insure its proper operation. If an internal malfunction occurs, the processor halts, all external outputs are turned OFF, and the HALT LED illuminates. Any other non-internal failure will cause the HALT LED to *flash*.

OFF: The processor is operating in the RUN, RUN/PROGRAM or DISABLE OUTPUTS mode (depending on whether the RUN LED is ON continuously or flashing).

FLASHING: If the HALT LED is flashing, one of the following conditions has occurred:

- The key selector switch has been placed in the HALT position.
- The user program or programming device has instructed the processor to halt. In this case, the light will continue to flash until the HALT bit is cleared.
- A memory or I/O error has occurred in which case either the MEMORY or I/O light will also be ON.

If needed, the processor can be forced into the HALT mode by the ladder diagram program or by the correct sequence of keystrokes on a programming device by setting bits 1 or 3 of control register 8176 within the processor. These bits are called the HALT bit and the HALT/RUN bit respectively. For detailed information on these bits, refer to Section 13.

• MEMORY

ON: The processor has detected an error in the user memory. This condition can be caused by trying to run the processor with no ladder program or system addressing in memory, or if a memory fault is detected in the ladder program. When the MEMORY LED is ON, the HALT LED will also be flashing.

OFF: User memory is operational.

FLASHING: Not used.

• FORCE

ON: Indicates that one or more inputs or outputs have been forced to an ON or OFF state, and that forcing via the COMM port (channel 2) is *disabled* (control register 8176 bit 5=1). Forcing overrides the actual input status or output commands of the ladder program. This forcing may have been via either the programming equipment or communication system. See Section 5 for information on the communication ports.

OFF: No I/Os are being forced.

FLASHING: Indicates that one or more inputs or outputs have been forced to an ON or OFF state, and forcing is *enabled* for the COMM port (control register 8176, bit 5=0).

• I/O

ON: A malfunction has occurred within the processor's internal I/O registers, or within the external input/output system. When the I/O LED is ON, the HALT LED will also be flashing.

OFF: Internal and external I/O systems are operational.

FLASHING: Not used.

• BATTERY LOW

ON: The lithium backup battery inside the Model 400 is low, and needs to be replaced. See Section 12.1 for battery information.

OFF: Both the onboard lithium battery *and* the power supply backup batteries are OK.

FLASHING: The power supply batteries are low, and must be replaced. If *both* the lithium battery and the power supply batteries are low, the steady ON condition prevails. This condition also occurs if any cable other than a Square D Type CC-10 cable is used to supply the Model 400 with power.

• WRITE PROTECT

ON: All or part of user memory is protected from alteration. For this LED to be ON, either the keyswitch is in the RUN position or the processor's internal write-protect jumper is in the NO PROGRAM position. See Section 6 for information on Memory Security features.

OFF: No security locks are active.

FLASHING: Some type of *software* security is in effect. See Section 6.

5 COMMUNICATION PORTS

5.1 Description

Two 9-pin RS-422 serial differential communication ports labeled PRGMR CHNL 1 (front panel) and COMM CHNL 2 (bottom) are provided on the Model 400. These ports allow connections to programming equipment and other peripheral devices.

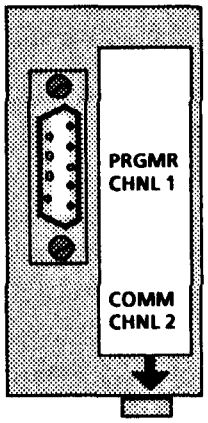


Figure 5.1 Communications Ports

The PRGMR and COMM ports work identically except that the PRGMR port allows *complete* access to the Model 400's systems while the COMM port can be set up to lock out certain functions. See Section 6.5.

Each port supports block READ/WRITE commands for a maximum count of 128 registers. Both ports also have an automatic timeout feature that monitors communication integrity when using READ, WRITE, or ALARM statements.

Both ports also support both ASCII input and output communication; refer to Section 10.

Immediate Communication Updates are supported for both ports; refer to Section 5.6.1.

The pinout for both RS-422 communication ports is shown in Figure 5.2

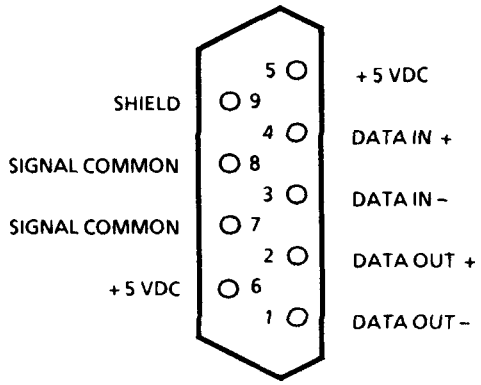


Figure 5.2 PRGMR and COMM Port Pinout

5.2 Connecting Programmers

Both ports will support all SY/MAX programming devices, including the Class 8010 Hand-held *Programmer. Since the Hand-held Programmer* obtains its power from the Model 400, its cable is limited to six feet (2 meters) in length. Other SY/MAX programming devices have their own power supplies, and can be placed up to 10,000 feet (3050 meters) from the Model 400.

Refer to the appropriate programmer Instruction Bulletins for more details on connecting programmers to the Model 400.

5.3 Connecting Other SY/MAX Devices

Many different peripheral or programming devices can be connected to the communication ports, greatly expanding the Model 400's capability and range of *applications. Figures 5.3 and 5.4 on the following* pages give some possibilities for peripheral connections to each of the two ports.

Both communication ports on the Model 400 support any device that uses the SY/MAX serial protocol. This includes, but is not limited to, the following devices:

- Loader/Monitor
- Other SY/MAX Processors
- D-LOG Modules
- Microcell Controller
- SY/NET Network Interface Modules
- Cartridge Tape Loader
- Printers
- Speech Modules

Connectable devices are described in this section. Since the *only* difference between the two ports is in security access (Section 6), the devices shown in Figures 5.3 and 5.4 can be connected to either communications port.

5.3.1 OTHER PROCESSORS

Other SY/MAX Processors (Types SCP-1XX, 3XX, 5XX, or 7XX) can be daisy-chained together, allowing them to share information and to function as an independent distributed control system. See the specific programmer's Instruction Bulletin for more information.

5.3.2 SY/NET NETWORK INTERFACE

The SY/NET Network Interface Module (NIM) is a local area communications network module capable of interfacing the Model 400 to an entire network of other programmable controllers, computers, programmers, etc. Refer to NIM Instruction Bulletin 30598-257-XX for more information.

NOTE: *The Model 400 can use variable ROUTE statements via Control Registers 8097 and 8098. See Section 5.5 for details.*

5.3.3 LOADER/MONITOR

The Loader/Monitor provides a simple and inexpensive method of monitoring I/O status and register values. It also allows register values to be changed without having to access the ladder diagram program.

The Loader/Monitor can display alphanumeric messages programmed in Model 400 memory, providing an inexpensive yet powerful method of signaling alarms, producing reports, and logging data. Refer to the Loader/Monitor Instruction Bulletin (30598-163-XX) for more information.

5.3.4 CARTRIDGE TAPE LOADER/RECORDER

By attaching the Cartridge Tape Loader/Recorder to the Model 400, the user's program can be stored on tape for backup purposes. Then if the user program inside the Model 400 is somehow altered or lost, the taped copy can be easily downloaded into the Model 400 as a replacement. Refer to the Cartridge Tape Loader Recorder Instruction Bulletin (30598-162-XX) for more information.

5.3.5 PRINTERS

Connecting a printer allows messages generated by the Model 400 to be recorded on hard copy. A printout ensures a permanent record of any received alarms or data. The recommended word structure for printers and other ASCII devices is to set the receiving device for 7-bits with odd parity and two stop bits. Alternate word structures are available via registers 8099 and 8100 (see Section 13.2). Refer to the Printer Instruction Bulletin 30598-176-XX or 30598-180-XX for more information.

5.3.6 D-LOG DATA CONTROLLER MODULE

The D-LOG gathers data from the Model 400 processor to generate documentation such as reports, alarm messages, and graphic displays. Refer to the Data Logger Instruction Bulletin 30598-272-XX.

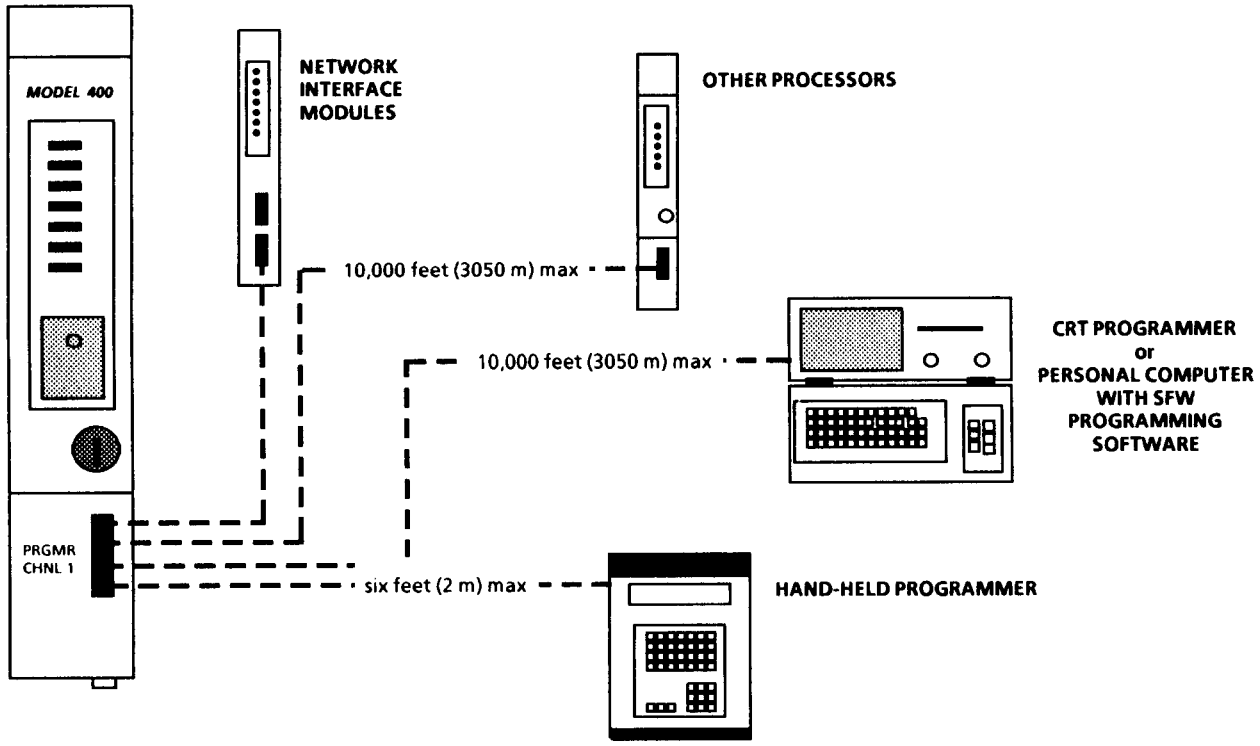


Figure 5.3 Possible PRGMR Port (Channel 1) Connections

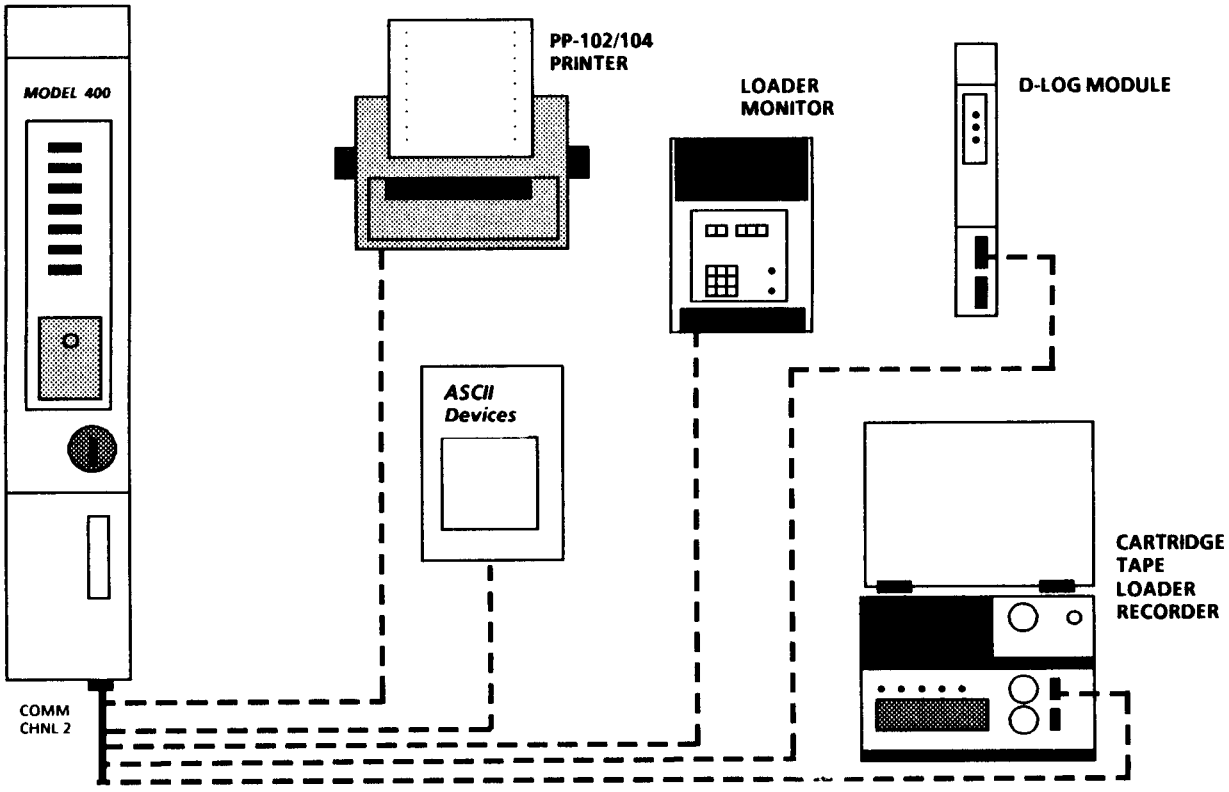


Figure 5.4 Possible COMM Port (Channel 2) Connections

5.4 Baud Rates

5.4.1 DESCRIPTION

The default communication rate of both ports is 9600 baud, which is the data rate used when the Model 400 is connected to most peripheral and programming devices.

Several methods are available to change the default baud rate should it be necessary for a Model 400 port to operate at other than 9600 baud. See Figure 5.5.

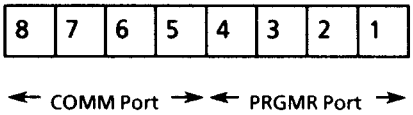
NOTE: *The BAUD rate can also be changed by using the DATA ENTER mode of a CRT, the SFW Programming Software, or a Hand-held Programmer. Enter the desired code into Model 400 control register 8169.*

OPERATION	EFFECT ON PRGMR PORT	EFFECT ON COMM PORT
PRINT command that specifies baud rate	Changes baud rate	Changes baud rate
Change the contents of register 8169 using a LET command, or while in the programmer's DATA ENTER mode	Changes baud rate	Changes baud rate
DEL/CLEAR ALL command	No effect	No effect
Power up	Set to 9600 baud	No effect unless memory is corrupt. If yes, set to 9600 baud.

Figure 5.5 Altering Baud Rates

5.4.2 ALTERING BAUD RATES

Control register 8169 in the Model 400 contains the bit patterns which determine baud rate for the communication ports. Bits 1-4 in register 8169 represent the baud rate for the PRGMR port, while bits 5-8 contain the baud rate for the COMM ports. Bit patterns are shown in Figure 5.6.



Control Register 8169 (Bits 1-8 shown)

The bit patterns corresponding to the various baud rates for both communication ports are shown in Figure 5.6 below.

The binary pattern for the PRGMR port is reflected in bits 1 to 4, while the pattern for the COMM port is in bits 5 to 8.

BAUD RATE CODE		BAUD RATE
Binary	Decimal	
0000	0	75
0001	1	110
0010	2	300
0011	3	1200
0100	4	2400
0101	5	4800
0110	6	9600
0111	7	19.2K

Figure 5.6 Baud Rate vs Bit Pattern

Baud rates can also be changed via word attribute registers 8099 and 8100. See Section 10.

5.5 Variable ROUTE Statement

5.5.1 DESCRIPTION OF OPERATION

The Model 400 can use variables to define routing for a communication rung. The format of the variable communication rung is similar to a rung that has a constant route, except that a special route - **205** - is inserted just after the regular route identifier in the rung. Two registers, 8097 and 8098, are used in conjunction with the variable route.

The generalized rung format for the variable ROUTE statement is shown below in Figure 5.7.

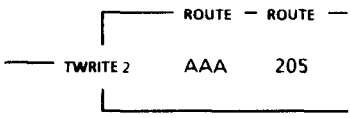


Figure 5.7 - Variable ROUTE in a COMM Rung

When programming the communication rung that is to have the variable ROUTE, the first ROUTE in the rung (AAA) should identify the originating device. The second ROUTE must be programmed with the special number 205. The number 205 causes the Model 400 to examine either register 8097 (if the PRGMR port is used) or 8098 (if the COMM port is used) for ROUTE data.

Register 8097 or 8098 act as pointers to a block of from two to eight registers. The user must load the register number of the first register of this block into register 8097 or 8098; each register in the block must then be preset with the intended route.

When the Model 400 executes the communication rung and encounters special ROUTE 205 as the message destination, it will check register 8097 or 8098 to find out where the variable ROUTE information is stored. The registers being pointed at by 8097 or 8098 must contain the intended routes.

The block of registers containing the ROUTE information will vary in length, depending upon the need for net-to-net routing. Since the length is variable, loading the value -1 (FFFFH) into a register defines the end of the block. In the majority of cases (no net-to-net used), the block will consist of only two registers – the first contains the ROUTE itself while the second (containing -1) identifies the end of the block.

The user must manipulate the contents of the first register in the block to obtain the variable ROUTE. If net-to-net routing is to be used, the size of the register block simply expands to accommodate the extra ROUTE information needed. Not until -1 is encountered in a register will the Model 400 assume that all intended routing information has been provided.

Thus, since SY/MAX protocol allows for messages to accept net-to-net routing information that is up to eight levels deep, the maximum block size is seven "route registers" plus the last (terminator) register that contains -1, up to a maximum total block size of eight registers. (Note that the originating route utilizes one of the eight levels.)

5.5.2 IMPLEMENTATION

The following steps are required to use the variable ROUTE feature in the Model 400.

1. Program a communication (COMM) rung normally, entering the number 205 as the destination ROUTE. Note that this also applies to communications that require multiple routes.

2. Prior to solving the COMM rung TRUE, load the starting register number of a block of up to nine registers into register 8097 (for the PRGMR port) or 8098 (for the COMM port).

3. Also before solving the rung, preset the "route register" block with the desired routes. Signify the end of the block by loading -1 (FFFFH) into the register that immediately follows the register containing the final destination route..

4. Reroute the communication message by changing the routing specified in the contents of the register block .

5.5.3 APPLICATION CONSIDERATIONS

- The maximum count in a Model 400 communication rung is 128 registers.
- The device communicating with the Model 400 may be unable to service the full 128 registers, in which case a transmission error occurs. Following is a list of register capacities for various SY/MAX devices:

Model 100	16
Model 300	16 (15 for Series A)
Model 400	128
Model 500	128 (Series M and later), 64 (Series L and earlier).
Model 600	128
Model 700	64
D-LOG Module	128
MicroCell Controller	128

- The communication statements used with variable ROUTE information are transition-sensitive.

5.5.4 EXAMPLE

Assume that the contents of registers 61 through 70 will be transferred *from* a networked Model 400 at location 101 *to* registers 91 through 100 in one of two processors, A and B. Processor A is at location 003. Processor B is on a separate network, at location 102, via a net-to-net link. The net-to-net link is at location 009.

The configuration of this example is shown in Figure 5.8 below. Note that Network Interface Modules (NIMs) are used with each processor.

NOTE: ***All NIMs in this example must be set to the SY/MAX protocol operating mode, except for the NET-TO-NET link units.***

The steps in Section 5.5.2 are used to configure the example.

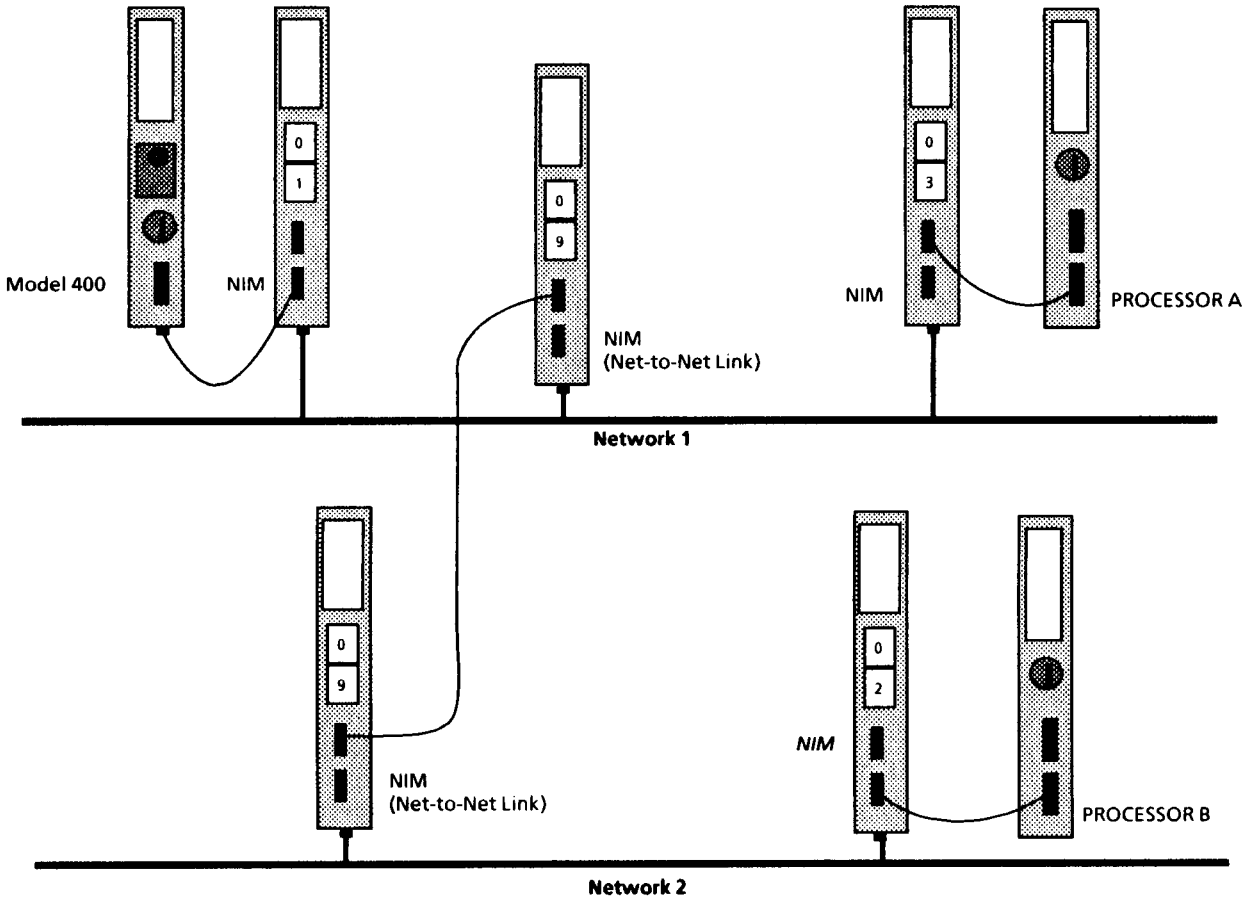


Figure 5.8 Variable ROUTE Example Configuration

EXAMPLE PROCEDURE

1. Program the rung shown below in Figure 5.9 in the **initiating** Model 400

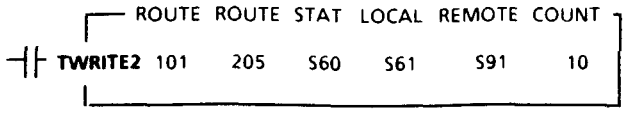


Figure 5.9 Rung for Variable ROUTE Example

NOTE: *The assigning of register S60 as the status register is arbitrary for this example. Any unused storage register can be used.*

2. The next step is to load the variable ROUTE information into the Model 400. Do this by choosing a starting register for the "route register" block; in this example, register 3901 is used. The Model 400 uses register 8098 (COMM port) for storing this variable ROUTE information; therefore, load 3901 into register 8098. Use the DATA ENTER mode of a programming device, or a LET rung within the Model 400's ladder program, to accomplish this.

3. Assuming that we first wish to transfer registers 61 through 70 to processor A, the value 0003 (processor A's location) must be loaded into register 3901. To terminate the block, load the value –0001 into register 3902 to show the Model 400 the end of the "route register" block. Again, a programming device in the DATA ENTER mode or a LET rung is used for this.

   When enabled by the logic, the communication statement is transmitted out of the Model 400's channel 2 (COMM) port through the NIM which is set as location 101.

When the logic in this communication rung is enabled, the Model 400 executes the rung with the following results:

- The ROUTE 205 in the rung causes the Model 400 to interrogate register 8098 for ROUTE information.
- Register 8098 (containing the value 3901) directs the Model 400 to examine register 3901.
- Since register 3901 = 3, the value 003 becomes the ROUTE destination.
- The Model 400 next interrogates register 3902. Since 3902 = –1, the Model 400 knows it has all the necessary ROUTE information. The contents of registers 61 through 70 are now sent to processor A's registers 91 through 100.

4. Now assume the register block 61 to 70 is to go to processor B's registers 91 through 100 instead. Since transmission to processor B involves a net-to-net link, an additional ROUTE parameter is needed. Enter the following values into the indicated registers:

   Register **3901** ..... **0009** (net-to-net link)

   Register **3902** ..... **0102** (processor B's location)

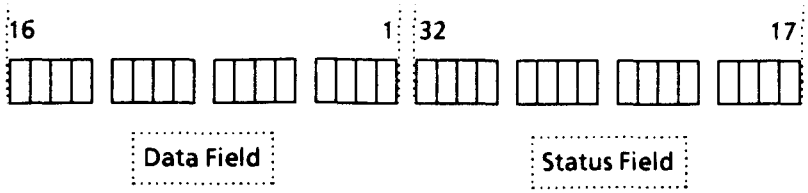
   Register **3903** ..... **–0001** (end ROUTE information)

   When the logic in the communication rung is enabled, registers 61 through 70 in the Model 400 are now routed through location 0009 to processor B at location 102.

5.6 Miscellaneous Considerations

Because the primary function of a programmable logic controller is the timely execution of the user program, communication with external devices is subject to certain rules and limitations. The following is a discussion of some of the potential pitfalls associated with these rules, as well as some techniques for improving communications throughput.

Both the sending and receiving of communication characters in READ, WRITE, ALARM, and PRINT (ASCII input/output) statements can occur at any time while the processor is scanning. Typically, the impact on scan time for servicing these characters is slight because of separate microprocessors used for control and communications. The largest single delay associated with the communication process occurs when a complete message has been received, and the register data contained in that message must be incorporated into the processor's image table registers. From 2.5 to 5.0 msec is allowed for the processor at the end of scan (EOS) to perform this task before scanning must resume. Note, processors which are in HALT can still respond to incoming messages, but cannot initiate them.



Bits	Description
1-14	Error Codes (Always Odd - bit 1 set)
15	Remote Device in RUN
16	Successful Completion
17	Message Sent to Output Buffer
18-21	SY/MAX Timer Value
22	*Input Ladder Logic Solved True*
23	SY/MAX Timeout Occurred
24-32	Register Identification Code (N/A)

Note: *Bits 1-23 are cleared when bit 22 is reset (input logic to Communications rung opens up)*

Figure 5.10 Communication Rung (READ, WRITE, PRINT, ALARM) SY/MAX Status Register

5.6.1 IMPROVING COMMUNICATION THROUGHPUT

The maximum block size of registers (count) which a user may specify in a message is 128 registers. For *communicating a large quantity of registers*, throughput is optimized by transmitting the largest blocks possible; a single transmission with 128 registers is over 10 times faster than 128 transmissions of 1 register each. The larger the message block, however, the more time is required at EOS to incorporate this data into the processor's image table. Because this processing time for 128-register messages can exceed 5 msec, multiple scans may be required to update all the registers. This presents a potential application problem if the new data needs to be treated as a contiguous block for control purposes. A simple example is floating point data, in which an odd/even pair of registers constitutes one floating point value, and splitting of this floating point register pair could result in an invalid value used by the PLC.

If the user desires to treat a block of registers contiguously, application programming techniques must be used to inhibit control logic operation until all the new data has been incorporated into registers. In the initiating processor, bit 16 of the associated communication rung status register is set when an operation is complete; thus, multiple bit 16's may be monitored to ensure that each associated message has been completed. Refer to Figure 5.10 for an overview of bit descriptions. However, when unsolicited WRITE commands are initiated by a remote device, the local device is not advised when an operation is complete.

Therefore, the remote device should use some of the transmitted registers to advise the local device. One technique is to increment the first and last register of the contiguous block each time a block of messages are sent; the local device would then need to be programmed to only act upon the data when these

registers were equal, indicating that the processor is not in the middle of an update and the data can be considered valid.

Another way of improving communications throughput by trading communication servicing time for scan time is the use of immediate communications update coils in the user program. Each time an immediate communications update coil (bits 8176-11 and 8176-12 for ports 1 and 2, respectively) is encountered, the processor checks the indicated port for a received message. If one exists, it will be serviced for up to 5 msec, thus imposing a worst-case scan time impact of 5 msec. For applications in which it is desirable to trade scan time for improved communications throughput on port 1, for example, coil 8176-11 may be used as often as desired at multiple program locations to process existing messages.

5.6.2 SY/MAX COMMUNICATION TIMEOUT

As discussed in the programming bulletin, bits of the communication status register (which must be uniquely assigned to the communication statement) can be used to monitor the communication process. For example, bit 22 is set to 1 when the input logic to the rung is true. Bit 17 is set to 1 when the message has been loaded into a commmunication buffer and is about to be transmitted. And bit 16 is set to 1 when a reply is received indicating the message transaction is successfully completed. (Bit 1, along with some combination of bits 2 through 14, is set to indicate an error and indicate an unsuccessful message transaction). Finally, bit 15, when set, indicates the remote device is actively scanning logic or executing a program.

Another feature available to the user is a communication reply timeout flag. This timeout is based on message *completion*, not just message acknowledgment. For a READ statement, for example, timing begins when the message is loaded into a communication buffer (bit 17 set to 1) and ends when the requested data has actually been received (bit 16 set to 1) or when time has expired or an error reply was received (bit 1 set to 1). Bit 23 of the communication status register is set any time a complete message response is not received prior to a user-specified period after the message has been initiated. This period is determined by the bit pattern contained in the appropriate bits of register 8168 at the time the communication rung is first solved true. If the time period selected in register 8168 is too short, the reply could still arrive after the timeout (bit 23) is set and the appropriate registers updated.

8168 Bits 1-4, 5-8 and 9-12:

BIT PATTERN (bits 4-1 for PRGMR bits 8-5 for COMM bits 12-9 for ETHERNET)	*TIME COUNT IN* MILLISECONDS
0000	0 to 250
0001	250 to 500
0010	500 to 750
0011	750 to 1000
0100	1000 to 1250
0101	1250 to 1500
0110	1500 to 1750
0111	1750 to 2000
1000	0 to 2000
1001	2000 to 4000
1010	4000 to 6000
1011	6000 to 8000
1100	8000 to 10000
1101	10000 to 12000
1110	12000 to 14000
1111	14000 to 16000

Figure 5.11 Communication Timeout Ranges

As can be seen from Figure 5.11, two ranges are available: 0 to 2 seconds (with 250 msec resolution) or 0 to 16 seconds (with 2 sec resolution). Note the time period associated with a bit pattern is an interval, with the timeout liable to occur anywhere within the interval. For example, a bit pattern of "0000" can result in a timeout being flagged anywhere from 0 to 250 milliseconds (maximum) after a message has been sent to the buffer.

When a message is initiated, the user-specified timeout code for that channel is copied into bits 18 to 21 of the corresponding communication status register. These bits then "count down" the time until a reply is received. If a reply is not received (bit 16 or bit 1 set to 1) within the timeout period, bit 23 will be set.

Note: *Each time the comms rung is solved false, all bits associated with the commmunication status register are cleared.*

COMM STATUS BIT DEFINITIONS:

Bit 16 only is set: A valid reply has been received; message complete *before* timeout has elapsed.

Bits 16 and 23 are set: A valid reply was received *after* the timeout had elapsed.

Bit 23 only is set: No reply received at all, timeout has elapsed.

Bits 1 and 23 are set: A valid *error* reply was received either before or after timeout. Check the data field for the error code.

CONSIDERATIONS FOR USING TIMEOUT

- Bits 1-23 in the status register are cleared every time the rung is executed FALSE.
- If a message is not acknowledged (no reply), ERROR 13 is posted in the status register and a processor communication port error is sent to register 8175 and bit 23 is set. For ports 1 and 2, this may occur when cabling to the ports is disconnected.
- If an error reply is received, the appropriate error code is posted in the status register and bit 23 is set, regardless of the actual time elapsed.
- If a valid reply is received after a timeout has occurred and bit 23 is set, bit 16 is set and bit 23 is latched ON until the communication rung is executed FALSE.

EXAMPLE: A user wishes to know if a certain ladder-initiated message transmitted over port 2 is receiving a reply within four seconds after initiation. The bit pattern **1001** *should be loaded into register 8168 bits* 5 through 8 prior to the communication rung being solved true.

If a reply is not received within 4 seconds (4000 milliseconds - the upper timeout limit), bit 23 of the appropriate status register is set. Keep in mind the bit *could* be set anytime after 2 seconds has elapsed since this is the lower timeout limit.

5.7 Technical Data For Communication Ports

TECHNICAL CHARACTERISTIC	APPLICABLE PORT		
	1	2	3
Connector Type			
D-type, 9-pin female slide lock (RS-422)	X	X	
ThinWire Ethernet- BNC, with gold-plated conductor			X
Communication Method			
RS-422 (serial differential)	X	X	
High speed Ethernet			X
Communication Rates			
(See Figure 5.5 for BAUD rate information)	X	X	
10 megabits/sec			X
Maximum Cable Length			
10,000 ft. (3050m) at 9600 baud (RS-422)*	X	X	
Ethernet-maximum:** 607 ft. (185 m) per segment, 3034 ft. (925m or five 185m segments) with repeaters			X
Cable Type			
Belden® #8723 or equivalent (RS-422)	X	X	
Ethernet-type RG58A/U or RG58C/U 50-ohm coaxial cable			X
Short Circuit Protection			
Driver/receiver circuitry	X	X	
Ethernet			No
Surge Protection			
Driver/receiver circuitry	X	X	
Ethernet			X
Common Mode Voltage Rating			
6 volts maximum	X	X	N/A
Auxiliary Power			
Powers the hand-held programmer or loader/monitor	X	X	N/A

* *Actual cable length for RS-422 communication is subject to installation considerations such as reliable building ground (signal reference), common mode voltage, signal interference/shielding, etc. If communication problems are experienced over long cable runs, decreasing the baud rate may result in greater signal integrity.*

** *Applies to 10BASE2 ThinWire Ethernet networks. Longer distances are possible using repeaters with other media such as 10BASE5 Standard Ethernet.*

Figure 5.12 Communication Ports Technical Data.

6 SECURITY FEATURES

6.1 Description

All four Model 400 processor types have security features that prevent unauthorized access to user ladder programs, labels, rack addressing and storage registers. This type of security protects programming equipment and/or other devices connected to the processor.

Three types of security features are used to protect each of the following:

1. Ladder memory
2. Data storage registers
3. Communications

Section 6.3 contains three tables that summarize all of the available security features in the Model 400 processor.

6.2 Definitions

The following terms are used frequently in this section.

Editing: The use of external equipment to insert, replace, delete or clear rungs, rack addressing, and labels that exist in the processor memory.

Register: Storage of I/O status, data values, or control information. Can be either *data* registers or *control* registers.

Non-Priority Register Read: Access to registers through programming equipment to display or copy (read-only) a value stored in a Model 400 data register.

Non-Priority Register Write: The use of programming equipment to enter or store a new value in a Model 400 data register.

Priority Register Read: The use of *non*-programming devices to recall a Model 400 register's contents. Typically, this type of communication occurs between two or more processors.

Priority Register Write: The use of *non*-programming devices to write or store a new value into a Model 400 data register. Typically this type of communication occurs between two or more processors.

Programming Device: SY/MAX CRT (SPR-200, 250, 300) SY/MAX hand held programmer (SPR-100) , loader monitor (SLM-100), or an IBM-Compatible computer equipped with SY/MAX Type SYM Programming Package.

User Memory: The memory in which user-created ladder programs, rack addressing, *and labels are stored.*

Storage Register Memory: Memory where data register values are stored in the Model 400, separate from user memory.

6.3 Summary

The three tables on the next page describe the ladder, register and communication security features available in the Model 400 processor:

- Figure 6.1 covers ladder program security features.
- Figure 6.2 covers storage register security features.
- *Figure 6.3 covers communication security* features.

In the following tables, the *method of protection* is indicated in the left column. *What the method protects is shown in the center column.* The right column gives the *subsection of Section 6* where that protection feature is described.

SECURITY METHOD	PROHIBITS	SEE SECTION
Hardware Jumper	Program editing of *any kind.*	6.4
Port/Route Edit Lockout	Program editing by any port or route other than the one used by the initiating equipment.	6.5
Keyswitch RUN position	Program editing of any kind, including forcing.	6.6
Inhibit Rung Coil	Altering rungs within an area defined by the inhibit rungs.	6.7
Inhibit/Safeguard Rung	Displaying, by programming equipment, the ladder areas defined by the Inhibit Rungs.	6.8
Password/ Restriction Register	Editing on a port-by-port basis.	6.9
Memory Protect Bit	Ladder programming or editing*	6.10

* Does not apply to PRGMR port

Figure 6.1 Ladder Program Security Methods

SECURITY METHOD	PROHIBITS	SEE SECTION
Password and Restriction Registers 8177 and 8178	• External non-programming devices from altering registers on a port-by-port basis. • Programming equipment from altering register data on a port-by-port basis • Forcing of I/O by programming equipment on a port-by-port basis.	6.9
Protect All Registers Bit	Altering of register data by any external device*	6.12
Force Inhibit Bit	Forcing of I/O by programming equipment*	6.11
Inhibit/Safeguard Rung	Programming rungs containing protected registers	6.8
Fence Registers	Altering of registers that are defined by user as fenced*	6.13

* Does not apply to PRGMR port

Figure 6.2 Register Data Security Methods

SECURITY METHOD	PROHIBITS	SEE SECTION
Inhibit/Safeguard Rung	The programming of COMM rungs (READ, WRITE, PRINT, ALARM) assigned to user-specified channels on a port-by-port basis	6.8

Figure 6.3 Communication Security Methods

6.4 Hardware Security Jumper

Prohibits: Ladder editing or programming of any kind through either port.

Uses: Internal jumper, located on a three-pin header. The jumper is accessed by removing the Model 400's side plate. Refer to Figure 6.4.

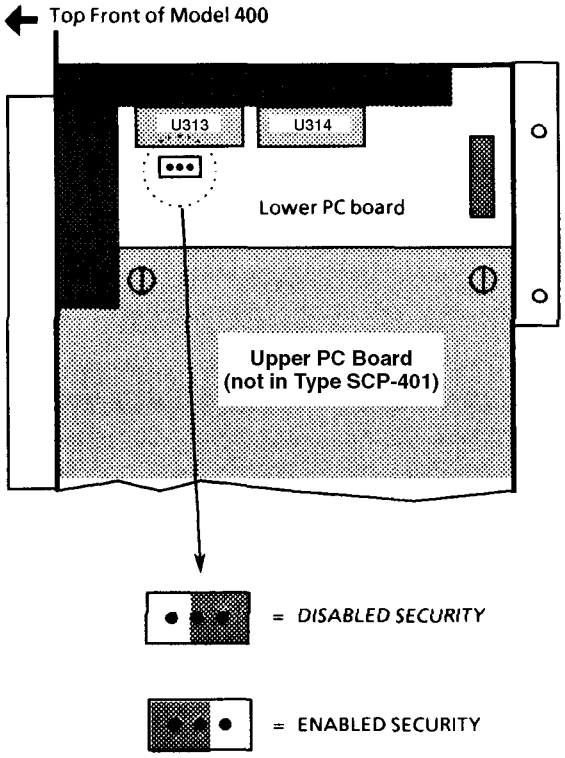


Figure 6.4 Hardware Security Jumper

The security jumper bridges the center pin of the three-pin header to *either* the left or right pin (see Figure 6.4). When the security jumper is in the ENABLED position, all ladder memory is protected from alteration of any kind. When DISABLED, ladder memory is accessible to the extent that other security methods allow. The WRITE PROTECT LED (Section 4.2) illuminates when the jumper is enabled.

CONSIDERATIONS

- The CLEAR ALL command will have no effect when the jumper is enabled.
- An enabled jumper overrides the following security features:

  1. Inhibit Rung (Described in Section 6.7)
  2. Memory Protect Bit (Section 6.11)
  3. Password & Restriction Register features for selective disabling of ports (Section 6.5)

SETTING THE SECURITY JUMPER

> **CAUTION**
>
> **Remove power from the rack before removing the Model 400 processor.**

1. Turn off the AC power to the rack where the Model 400 is installed.
2. *Remove the Model 400 from its rack.*
3. Remove the side access plate from the Model 400.
4. Using a needlenose pliers, place jumper in the desired position (ENABLED or DISABLED) as shown in Figure 6.4. DO NOT LEAVE THE JUMPER OFF OF THE HEADER. It must be in one of the two positions.
5. Replace the side cover on the Model 400, and reinsert it in its rack.
6. Reapply rack power.

6.5 Selective Port & Route Edit Lockout

Prohibits: Ladder program alteration through either port or by any route *except the port and route used by the initiating programming device.*

Uses: Control Register 8176, bit 15.

This security feature insures that only a single programming device on the SY/NET has access to the Model 400's memory at any given time. If SY/NET is not in use, the restriction applies channel-by-channel.

To engage the Port/Route Lockout feature, set bit 15 of Control Register 8176 to "1" using the Data Enter mode of the programming device. The Model 400 "remembers" the port and route that the programming device used to set bit 15. Any attempt to edit the program through any other port or route other than the "remembered" port/route causes the Model 400 to generate an ERROR 49.

The lockout feature does *not* prohibit simple interrogation or viewing of the ladder program, unless other security features that prevent these are active. *Register* contents can still be altered (unless other measures are active to prevent this). To release the lockout, use the programmer that set bit 15 to "1" (the same route/port) to reset the bit to "0".

CONSIDERATIONS

- A CLEAR ALL command issued through the PRGMR port of a Model 400 that has Port/Route Lockout enabled has no effect *unless* the device sending the CLEAR ALL is the same device that enabled the lockout.
- Cycling power to the Model 400 will disable the lockout and extinguish the WRITE PROTECT LED (if no other form of security is active).
- If the lockout is disabled by cycling power, bit 15 of Control Register 8176 may remain set at 1. This is because the processor retains Control Register contents, but *cannot* retain port and route information, when power is removed. Thus, the fact that bit 15 of 8176 is set at 1 is not a guarantee that the lockout feature is active. Conversely, if bit 15 is set to 0, the lockout feature may still be active. Only by setting the bit *immediately prior* to editing, and resetting the bit *immediately after* editing can bit indication be considered reliable.

6.6 Keyswitch RUN Position

Placing the Model 400 keyswitch in the RUN position prohibits any changes to be made in a ladder program. In addition, the ability to force, or to alter forcing, is disabled while the keyswitch is in RUN (if changes to a program that is running are to be made, place the keyswitch in the RUN/PROGRAM position).

6.7 *Inhibit Rung Coil*

Prohibits: Altering rungs within protected ladder areas that are defined by the Inhibit Rungs.

Uses: Control Register 8176, bit 16

A maximum of two areas of the user ladder program can be protected from being edited. The two areas that make up a typical ladder program are the *MAINLINE* ladder area and the *SUBROUTINE* ladder area.

The *subroutine* area consists of all the subroutines that exist in the user ladder program. This area starts with the MARK ST SUB rung and proceeds to the last rung in memory. The *mainline* ladder area consists of rung number 1 in memory and proceeds up to, but not including, the MARK START SUB rung.

Protected *mainline* ladder rungs can start with rung number 1 in the ladder and consecutively proceed up to the inhibit rung in the mainline area.

The protected *subroutine* rungs always start with the MK ST SUB rung and proceed consecutively to the inhibit rung in that area.

An inhibit rung is inserted into a program by entering a rung containing only a coil addressed as *register 8176 bit 16*. See Figure 6.5, below:

Figure 6.5 Inhibit Rung

CONSIDERATIONS

1. Protection is not enabled until either the processor is power-cycled or the keyswitch is turned from HALT to RUN.

2. More than one inhibit rung may exist in any given area, but only the earliest (lowest number) programmed inhibit rung in an area defines the end of protected ladder.

3. When a *CLEAR ALL* command is issued to the processor over channel 1, the protected areas defined by the inhibit rungs will *not* be cleared. Only the unprotected areas of the ladder program will be cleared.

CLEARING THE INHIBIT RUNG

Clearing the protected and unprotected areas of a ladder program that is protected by Inhibit Rungs requires that the *entire* ladder program and all rack addressing be cleared from the Model 400. This is done by connecting a programmer to the PRGMR (channel 1) port, and then performing the following steps:

1. Confirm that the security jumper (Section 6.4) is in the DISABLED position.

2. Access the RACK ADDRESSING mode.

3. Press the DELETE soft key.

4. Press the CLEAR ALL soft key *twice*.

5. Access the DATA ENTER mode.

6. Store the decimal value **-1** in register 8177 (the password register).

7. Repeat steps 2 through 6.

The result of the above CLEAR ALL operation is a completely blank user memory, meaning that:

- No Ladder Program
- Default Rack Addressing
- Zeroed Register Values
- No Labels
- No Enabled Security Features

...will exist within the Model 400.

When using the SCP-444 RAM/UVPROM processor, the above procedure copies the contents of the UVPROM memory into the RAM memory. Refer to *Appendix A for details.*

Figure 6.6, below illustrates the operation of the Inhibit Rung Coil.

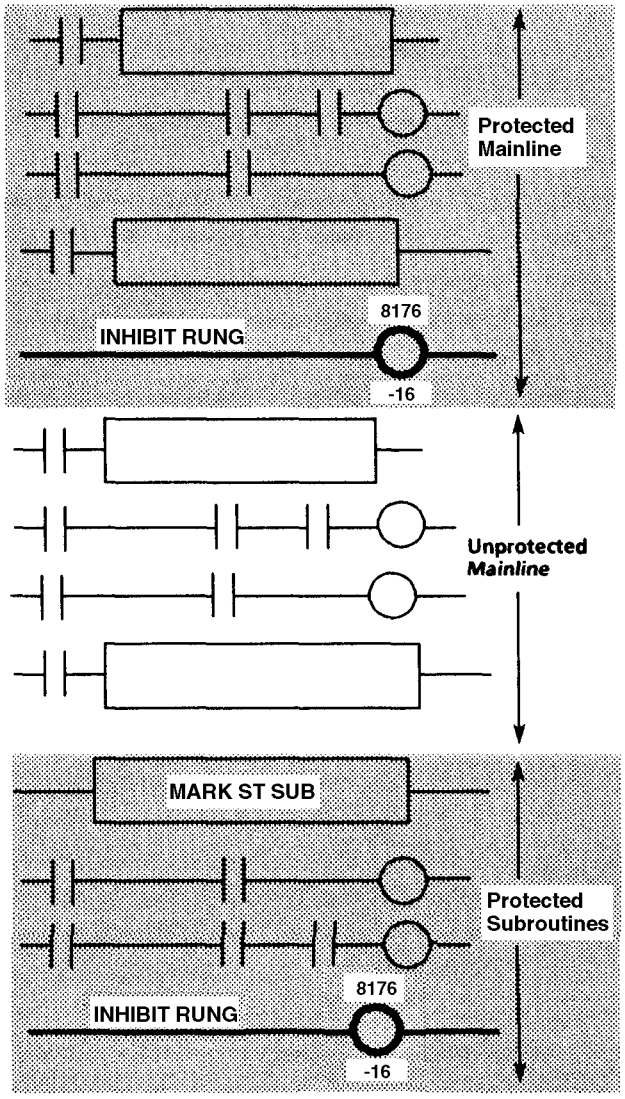


Figure 6.6 Operation of the Inhibit Rung Coil

6.8 Inhibit and Safeguard Rung

Prohibits:

1. *Displaying, by programming* equipment, any rungs protected by Inhibit Rungs (see Figure 6.6). Two different groups, each with an Inhibit Rung, can be protected.
2. The programming of any COMM rungs (READ, WRITE, PRINT, ALARM) assigned to user-specified channels on a channel-by-channel basis.
3. Programming any rungs that are user-defined as protected.

Uses:

1. Inhibit Rung with Safeguard Rung, Parameter A (#1 and 2 above).
2. Inhibit Rung with Safeguard Rung, Parameters B and C (#3 above).

The safeguard rung allows a variety of security protection features to be enabled. To use the safeguard features the special TLET rung as described below must be programmed as the first rung in ladder memory. To activate the **A, B_1, B_2, C_1, C_2,** and **D** parameters in the special TLET rung, an inhibit rung with coil addressed as 8176-16 must *also be programmed.*

The following rung **must be** rung #1:

—[TLET S8165 = A ; B_1 ; B_2 ; C_1 ; C_2 ; D ; E_1 ; E_2]—

Where:

A is port safeguard code for COMM rung/view inhibit.

B_1 is *STARTING* ADDRESS for Group 1 Safeguarded Registers.

B_2 is *ENDING* ADDRESS for Group 1 Safeguarded Registers.

C_1 is *STARTING* ADDRESS for Group 2 Safeguarded Registers.

C_2 is *ENDING* ADDRESS for Group 2 Safeguarded Registers.

D *is the SCAN TIME LIMIT in milliseconds*

E_1 is *TOLERANCE OF SCAN INTERRUPT* in 2.5 millisecond counts.

E_2 is *RATE OF SCAN INTERRUPT* in 2.5 millisecond counts

The parameters above are explained in detail below:

A A number entered here (000, 001, 002, or 003) prevents Channel 1 and/or Channel 2 ports from being assigned to communication rungs (READ, WRITE, PRINT, ALARM). Figure 6.7 shows which number restricts what port(s):

CODE	PRGMR Port Restricted	COMM Port Restricted
X00	NO	NO
X01	YES	NO
X02	NO	YES
X03	YES	YES

Figure 6.7 Port Safeguard Codes

When "1" is substituted for "X" as the first digit in the number (i.e., 100, 101, 102, or 103), the ability to view, display, record or print the rungs through both ports is also prohibited.

EXAMPLE: To prevent only the PRGMR port (channel 1) from being assigned to *communication rungs AND to inhibit the ability* to view, display, record, or print out the program through *either* port, enter the number "101" as the port safeguard code.

Safeguarding *registers* prohibits those registers from being programmed into counters, timers, or shift registers. Also, safeguarded registers cannot be used to the left of an equal sign in a LET statement, or have their bits programmed as coils.

Safeguarded registers are also protected from being written to by other SY/MAX processors *unless* the other processor's WRITE communication rungs are also protected from within an Inhibit Rung ladder area.

B_1 *Starting* address for the *1st group* of safeguarded registers. The starting address can be any value from 1 to 8192.

B_2 *Ending* address for the 1st group of safeguarded registers. This address must be greater than or equal to **B_1**.

B_1, **B_2**, **C_1**, and **C_2** *must* be separated by a softkey-generated semicolon.

C_1 *Starting* address for the *2nd group* of safeguarded registers. The starting address can be any value from 1 to 8192.

C_2 *Ending* address for the 2nd group of safeguarded registers. Must be greater than or equal to **C_1**

D *Scan time limit* in milliseconds. If the Model 400 exceeds the scan time limit, it will halt with "ERROR 970" in error register 8175. When **D** is "0", a default scan time limit of 1 second is assigned.

INHIBIT/SAFEGUARD RUNG CONSIDERATIONS

- Zeros should be placed in parameter locations for features that are unused. For example, if only one group of registers is to be protected, enter "0" for the C_1 and C_2 parameters.
- If Timed Interrupt tolerance and rate are the only parameters used (no Safeguard parameters), only values for E1 and E2 need be entered. Because the Safeguard rung parameters are read from right-to-left, E1 and E2 will be the first parameters encountered and are interpreted as Timed Interrupt parameters. Omitting values for other parameters causes them to be ignored.
- *Only* register 8165 can be assigned as the *safeguard TLET rung.*
- Only one safeguard TLET rung is allowed in the entire ladder program, and it must be the *first* rung in the program.
- Make sure the Model 400 is in HALT before entering the Safeguard rung into the program. If the Model 400 is in RUN while the rung is entered, the rung will execute normally and possibly create unwanted values in registers 8165, 8166, 8167, and any others used in the rung.
- *Parameters A;B1;B2;C1;C2 of the TLET rung* require that an inhibit rung exists in the ladder program. If no inhibit rung exists, the safeguard parameters A;B1;B2;C1;C2 are ignored.
- The safeguard *TLET* rung parameters cannot be altered once the inhibit rung is programmed and the Model 400 is power-cycled or is keyswitched to RUN. *All ladder program and rack addressing must be deleted before the safeguard rung can be altered.*
- If password register 8177 is included in the block of safeguarded registers, the only way to override and delete the safeguard rung is to remove the Model 400 from the rack, and then remove the onboard battery. This allows the memory to completely discharge.
- To override and delete the safeguard rung, the entire ladder program and all rack addressing must be cleared from Model 400 memory. This is accomplished by connecting programming equipment to channel 1 (PRGMR Port), and *performing the following steps:*

  1. Confirm that the Security Jumper is in the DISABLED position (Section 6.4).
  2. *Access the RACK ADDRESSING mode.*
  3. Press the DELETE soft key.
  4. Press the CLEAR ALL soft key *twice.*
  5. Access the DATA ENTER mode.
  6. Store the decimal value **-1** into register 8177 (the password register)
  7. Repeat steps 2 through 6.

  The result of the CLEAR ALL operation is a completely blank user memory, meaning that. . .

  - No Ladder Program
  - Default Rack Addressing
  - Zeroed Register Values
  - No Labels
  - No Enabled Security Features

  . . . . will exist within the Model 400.

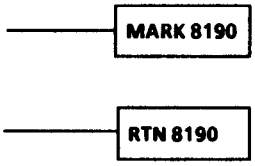
When using the SCP-444 RAM/UVPROM processor, the above procedure copies the contents of the UVPROM memory into the RAM memory. Refer to Appendix A for details.

6.8.1 TIMED INTERRUPT OPERATION

The parameters E_1 and E_2 in the TLET S8165 rung define the values for timed interrupt *tolerance* and timed interrupt *rate*, respectively.

This method of scan control interrupts the ladder program at specified time intervals to execute a certain group of rungs (a special subroutine), and then returns to the main program. This method of interrupt is useful for applications where certain inputs of data must be *continuously sampled within a particular repeatable time limit.*

Subroutine number 8190 is reserved for the timed interrupt function. The following two rungs are used to identify the timed interrupt subroutine:



Refer to the "Processor Scan Control" section in the respective programmer manual for an in-depth discussion of using GOTO, GOSUB, and TIMED INTERRUPT scan controlling techniques.

The *purpose* of the timed interrupt is to ensure that, regardless of program length, a specific block of rungs is executed within a given period of time. Typically this involves monitoring a select group of *inputs* while programmed rungs control a select group of *outputs*.

The timed interrupt subroutine should *not* contain any transition-sensitive rungs, or any rungs that require multiple program scans to complete their operation. Examples of these rungs include communication statements (READ, ALARM, WRITE), transitional coils or LETs (TLETs), shift registers, and timers.

While the programmed interrupt rate specifies the rate at which the interrupt is to occur, it may not always be possible to interrupt the Model 400's scan *precisely* at that rate. For example, the processor cannot be interrupted while it is scanning a rung or during end-of-scan activities such as updating external I/O and servicing communication ports.

NOTE: *RUN-mode programming is not allowed while the TIMED INTERRUPT is enabled; attempts will be rejected with ERROR 79 posted by the programmer.*

In order to be assured that the processor has sufficient time to call the timed interrupt subroutine, the programmed tolerance must exceed the external I/O update time plus the communication port servicing time. The time required to update external I/O is a function of system layout and rack addressing; refer to Section 14.2 for more information regarding I/O updating. Servicing the communication ports can require an additional 5 milliseconds per scan if saturated. These factors can combine to form a substantial time period during which the Model 400's scan *cannot* be interrupted.

CAUTION

A program containing a Timed Interrupt should ALWAYS be tested first, under conditions that simulate the Rack Addressing and communication demands that the system will experience when installed at the actual job site.

In the event that interrupt tolerance errors (ERROR 29102) are encountered, it may be possible to eliminate these errors by optimizing rack addressing and/or restricting communication traffic. If errors persist, the programmed tolerance value needs to be increased.

In the majority of applications that require speed, the inherently fast scan speed of the Model 400 eliminates the need for the timed interrupt function.

NOTE: *Since the Model 400 updates external I/O at the end of each ladder scan, any timed interrupt that is executed more than once per scan must also update the external I/O at the same rate as the timed interrupt occurs, as discussed in the following.*

Registers 8105 and 8106, in conjunction with bit 7 of register 8176, can be programmed to perform an immediate I/O update within the timed interrupt (see Section 13.2). This selectively updates any registers that require fresh information. Thus, registers used as *inputs* within the timed interrupt subroutine should be updated at the *beginning* of the subroutine, while *output* registers within the subroutine should be updated *just prior to exiting* the subroutine.

A scan penalty is paid to accomplish immediate I/O updating, and must be considered when determining the execution time of the timed interrupt subroutine. Refer to Figure 14.3. The time needed for this updating becomes more and more significant when the ladder portion of the subroutine is short. To economize these register updates, group together input and output addresses that are used in the timed interrupt to minimize the scan time impact of performing immediate I/O updates.

6.9 Password and Restriction Registers

The Model 400 processor uses register 8178 for storing an access code and register 8177 for storing a "password" value.

The following restrictions can be imposed on either port by using Password Register Restriction:

1. ***Inhibit ladder programming or editing.*** Restricts the programming or editing of ladder logic using any programming device. *Bit 4 or 8 of register 8178.*

2. **Prevent programming equipment from forcing I/O registers.** Restricts the forcing of register outputs using programming equipment. *Bit 2 or 6 of register 8178.*

3. **Prevent programming equipment from altering register data** . Prevents the using of programming equipment to write to or change data values that are stored in all registers. *Bit 3 or 7 of register 8178.*

4. **Prevent any non-programming devices from altering registers.** Prevents the use of processor-to-processor communication (priority communication) to alter data values stored in all registers. *Bit 1 or 5 of register 8178.*

The bit pattern in register 8178 (the *restriction code* register) determines which port (channel) is protected, and by which restrictions. Figure 6.8 illustrates which bit in register 8178 needs to be turned ON to achieve the desired restriction and port.

REGISTER 8178, *BIT # "ON"*	RESTRICTION AND PORT
1	PRGMR port -- processor-to-processor communications. Restricts writing data to Model 400 registers
2	PRGMR port -- forcing of I/O is prevented using programming equipment
3	PRGMR port -- programming equipment to processor communications. Restricts writing data to Model 400 data registers.
4	PRGMR port -- restricts altering the ladder program in any way.
5	COMM port -- same effect as bit 1
6	COMM port -- same effect as bit 2
7	COMM port --same effect as bit 3
8	COMM port -- same effect as bit 4

Figure 6.8 Restriction Code Bit Table

To enable the restriction code, a non-zero "password number" must be entered (via the DATA ENTER mode of a programming device) into register 8177.

Once a password value is entered into register 8177, a "**-1**" is displayed when attempts are made to monitor this register. This keeps the password confidential. Thus, "-1" *cannot* itself be used as a password value.

The restriction code can be altered only by the *exact* password value entered into register 8177. When this is done, register 8177 will display "**0**".

To *disable* the password/restriction code feature, enter a CLEAR ALL command through the PRGMR port. This is allowed regardless of the restriction code in effect, and resets the password register to *zero*. All *rungs not protected* by Inhibit Rungs are also deleted.

6.10 Memory Protect Bit

Prohibits: Ladder programming or editing through the COMM port (does not affect the PRGMR port).

Uses: Control Register 8176, bit #4.

When bit 4 of register 8176 is "ON", all user ladder program, rack addressing, and labels are protected from editing. This protection feature affects the COMM port (channel 2) only. It is ignored by the PRGMR port (channel 1).

To *enable* the Memory Protect Bit, two methods can be used:

1. Program a rung containing only a coil addressed as bit 4 of register 8176, like this:

2. With a programming device in the DATA ENTER mode, enter a "1" in bit 4 of register 8176.

To *disable* the Memory Protect Bit, delete the rung AND reset bit 4 to "0".

NOTE: *Merely deleting the rung does not turn bit 4 off, and by itself does not disable Memory Protect.*

6.11 Force Inhibit Bit

Prohibits: Forcing of I/O by programming equipment through the COMM port (does not affect the PRGMR port).

Uses: Control Register 8176, bit #5.

When bit 5 of register 8176 is "ON" the forcing of I/O registers (using programming equipment) through the Model 400's COMM port (channel 2) is prohibited. This feature is ignored by the PRGMR port (channel 1).

To *enable* the Force Inhibit Bit, two methods can be used:

1. Program a rung containing only a coil addressed to bit 5 of register 8176, like this:

2. With a Programming device in the DATA ENTER mode, enter a 1 in bit 5 of register 8176.

To *disable* the Force Inhibit Bit, delete the rung AND THEN reset bit 5 to "0".

NOTE: *Merely deleting the rung does not turn bit 5 off, and by itself does not disable Force Inhibit.*

6.12 Register Protect Bit

Prohibits: Altering of register data by any external device through the COMM port (does not affect the PRGMR port).

Uses: Control Register 8176, bit #6.

When bit 6 of register 8176 is ON, the altering of any Model 400 data registers through the COMM port is prohibited. This restriction feature is ignored by the PRGMR port.

To enable this feature, two methods can be used:

1. Program a rung containing only a coil addressed to bit 6 of register 8176:

2. Use a programming device in the DATA ENTER mode to enter a "1" in bit 6 of register 8176.

To *disable* the Register Protect Bit, delete the rung AND THEN reset bit 6 to "0".

NOTE: ***Merely deleting the rung does not turn bit 6 off or disable Register Protect.***

6.13 Fencing Registers

Prohibits: Altering of any registers within a user-defined fenced area through the COMM port (does not affect the PRGMR port).

Uses: Control Registers 8173 and 8174.

Registers 8174 and 8173 contain, respectively, the beginning and ending addresses of an *unprotected* user-defined block of data. All registers *outside of* the defined block are protected from alteration by outside devices. The default (power-up) value in register 8174 is "1"; for 8173, it's "8176". Thus, registers 8177 through 8192 are effectively fenced.

7 FLOATING-POINT MATH
(Not Applicable to Type SCP-401)

7.1 Introduction

This section provides information on Model 400 Processor Types SCP-423, 424, and 444 floating-point (FLP) math capabilities (SCP-401 works only with integer values). Floating-point math allows these Model 400 processors to perform high-level math operations on data values containing a floating decimal point.

The Model 400's floating-point operation conforms to the ANSI/IEEE Standard 754 pertaining to the 32-bit standard for floating-point math.

7.2 Definition of Terms

Compare (IF Instruction) Comparison of a value in one data register to a constant, to the value of a second register, or to the result of a math operation.

Intermediate Result .. The result of one math operation in an expression containing multiple operations.

Operator ..... Math functions + − × ÷

Overflow ..... *Exceeding the numerical* storage range of the data register.

Result ..... The *final* value of any mathematical expression (*result = A + B - C*).

Soft Key ..... Also called a *function key*. Multifunction keys located above the uppermost row on the *keyboard of a CRT* Programmer or function keys on an IBM® keyboard using SFW Programming Software.

Transfer (LET Instruction) *Storing a constant* numerical data value or math result in a data storage register.

7.3 Register Usage

The Model 400 processor uses different methods to deal with data stored in integer and floating-point registers. See Figure 7.1.

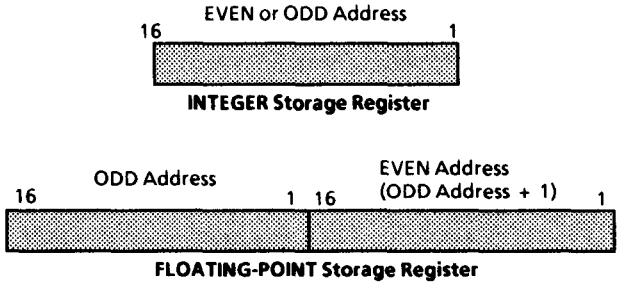


Figure 7.1 Floating Point vs. Integer Register

7.3.1 INTEGER

The range for an integer data value stored in an integer register (to be used in a math calculation) is ±32,767. The exceptions to the range limitations are registers used in timers, counters, and matrix pointers. Registers that store the current time and count can range from 0 to +9999, while matrix pointers cannot exceed 4,000.

7.3.2 FLOATING-POINT OPERATIONS

The Model 400 uses two consecutively addressed 16-bit data registers to create a 32-bit floating-point data register, as shown above in Figure 7.1.

A floating-point register can contain a number from $\pm 3.4 \times 10^{38}$ to $\pm 1.2 \times 10^{-38}$. The mantissa is limited to seven digits. Values are entered and displayed using scientific notation. This notation is shown as "E" followed by an integer representing the power to which 10 is raised and multiplied by the mantissa. For example, "25,000" is displayed as "2.5 E+04". See Section 7.5.

7.4 Addressing Floating-Point Registers

Of the register pair that make up a floating-point register, *only the ODD-numbered register can be used as the address.* If the *even* address is used, an "Illegal Address" error results. The Model 400 *automatically* assigns the even-addressed register that follows the odd-addressed register. See Section 7.5 for instructions on how to directly enter floating-point numbers.

When a rung containing a floating-point register is entered into a ladder program, the letter **F** (generated by the **FLP** soft key) precedes the odd address value of that register. Refer to the following example.

EXAMPLE: *Given* - A floating-point register pair, F0051/F0052, is used to store the value 0.001234567 (1.234567×10^{-3}). The rung used to program this information is shown in Figure 7.2:

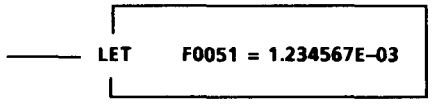


Figure 7.2 Example 1 Floating-Point LET Rung

Procedure: Press the **LET** soft key to produce another menu of soft keys.

This new menu of soft keys describes and defines the various data register types as shown below:

REG ..... Integer Register

MATRIX ..... Integer Matrix

FLP. REG ..... Floating-Point Register

FLP. MAT ..... Floating-Point Matrix

FLP. IMD ..... Floating-Point Constant

Since a floating point register is being used here, press the **FLP.REG** soft key and enter the *odd* register (0051) address. See Figure 7.3:

F0051	F0052
16 Bits	16 Bits

Figure 7.3 Example Register Structure

In Figure 7.3, F0051 specifies that register pair 0051 and 0052 will store the value 0.001234567 in 32-bit format.

Register F0052 *automatically* becomes a part of the floating-point register addressed as F0051. Once assigned as half of a floating-point register pair, register 0052 CANNOT be used for any other purpose in the same ladder program, since the bit pattern is only meaningful as part of a floating-point pair.

NOTE: *Attempting to program an even-numbered floating-point address results in an "illegal address" error.*

After entering the register address, complete the box using the "=" and "FLP.IMD" soft keys, followed by the desired floating-point number.

LADDER PROGRAM REGISTER USAGE

Any of the 4000 user-addressable registers in the Model 400 can be designated as floating-point registers. Since each 32-bit floating-point register requires an odd-even 16-bit register pair (32 bits), approximately 2000 floating-point registers can be used. See Figure 7.4.

While it is *not necessary* to dedicate a block of registers to contain floating-point values, it is always a good idea to group together registers that serve the same purpose such as external I/O, internal I/O, integer storage, and floating-point storage.

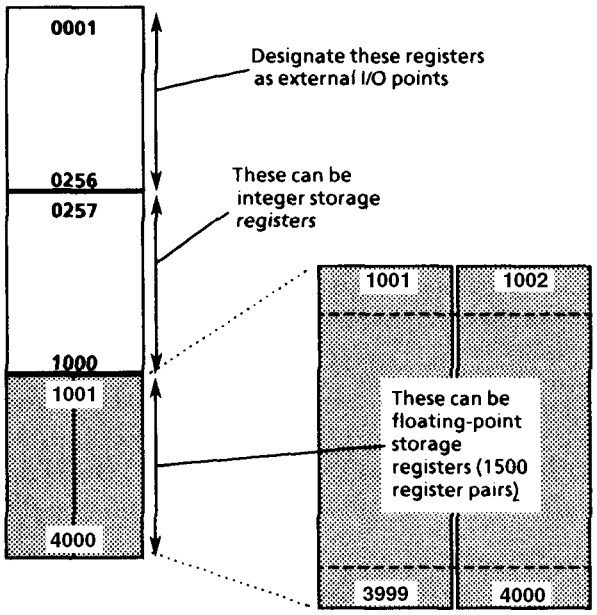


Figure 7.4 Sample Register Allocation

7.5 Data Transfers (LET) and Comparisons (IF) Using Floating-Point Numbers

Floating-point registers are allowed within two types of ladder program instructions:

1. **LET** Instructions
2. **IF** Instructions

7.5.1 DATA TRANSFERS ("LET" INSTRUCTION)

The LET operation copies a value (a constant, data value, or math result) from the right of the equal sign to the register(s) on the left side of the equal sign. Figure 7.5 shows the operation of an *integer* and a *floating point* data transfer (LET instruction).

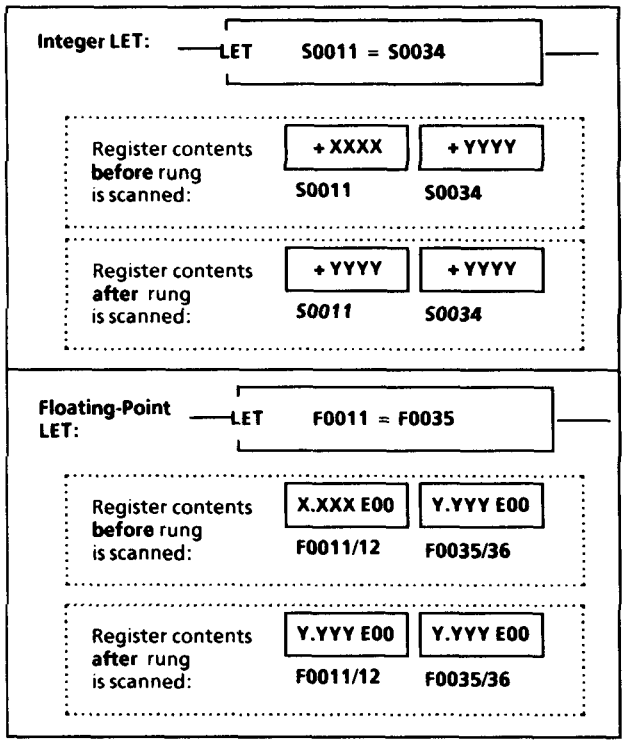


Figure 7.5 Operation of LET Instruction

See Section 7.7 for math operations allowed within LET and IF instructions.

7.5.2 COMPARISON ("IF" INSTRUCTION)

The IF instruction compares the contents of a an integer or floating-point register on the left of the COMPARE symbol to a constant or value stored in an integer or register to the right of the COMPARE symbol.

When comparing two floating-point values for equality, the two values must be *exactly* equal before the IF rung executes TRUE.

7.6 Entering Floating-Point Values

SY/MAX family devices that can enter floating-point data *directly* into floating-point registers include:

- Type SPR-300 Deluxe CRT Programmer, Series "i" (Revision 2.6 or later).
- SY/MATE Programming Software.

ENTERING FLOATING POINT NUMBERS VIA THE PROGRAMMING EQUIPMENT'S DATA MODE

To enter a floating-point value into a register, the keystrokes shown in the following table are required. Assume that the number 0.001234567 (1.234567×10^{-3}) is to be entered into odd-addressed register *0051*.

The [bracket] notation refers to a soft ("function") key, while the **bold** is a dedicated keyboard key(s).

Step #	Key(s) Used	Remarks
1	[DATA]	Accessed from STATUS screen.
2	[FLP.REG]	Accessed from DATA screen.
3	**51**	Enters the first address of the register pair.
4	[BIN.FLP]	When pressed, the cursor will move down one line and will show the even-numbered address (**0052**) of the selected floating-point register pair.
5	**1.23456E-3**	Enter the floating-point value.
6	[ENTER]	Loads the number into the register pair.

FLOATING POINT CONSIDERATIONS

- A seven-digit mantissa coupled to a double-digit exponent is the limit of accuracy when entering a floating-point number.
- A negative value can be entered by pressing the [=] key immediately after the [BIN.FLP] soft key is pressed.
- If a value is not entered in scientific notation, the Model 400 automatically converts the value to scientific notation when the [ENTER] soft key is pressed.
- When entering a value in scientific notation, a negative exponent is specified by pressing the [=] key after the [E] key.

7.7 Displaying Floating-Point Values

When using a programming device, two methods for viewing the data stored in floating-point registers are available: the Data Display Mode and the Ladder Rung Display Mode. Each of these is explained below, and illustrated on the next page.

METHOD 1: From the DATA mode, press the FLP.REG softkey. Next, enter the register's *odd-numbered* address, then press the DISPLAY softkey (see Figure 7.6). When using the Programming Device in the DATA DISPLAY mode, floating-point values are displayed in the BIN/FLP column in the *even* (F0052) register address row.

NOTE: ***The "decimal" column of the DATA mode screen displays meaningless data if the register is a floating point register.***

METHOD 2: Display the rung using the floating point register by searching for the rung, then pressing the DISPLAY soft key (see Figure 7.7).

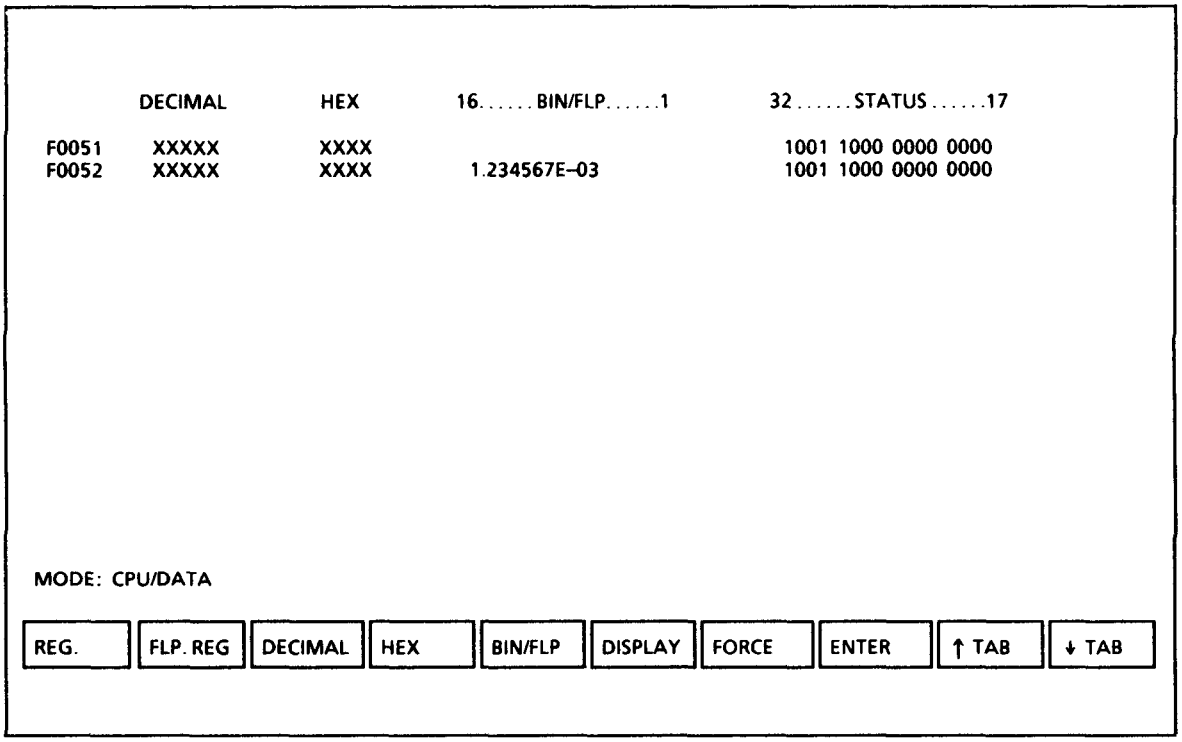


Figure 7.6 Data Display Mode

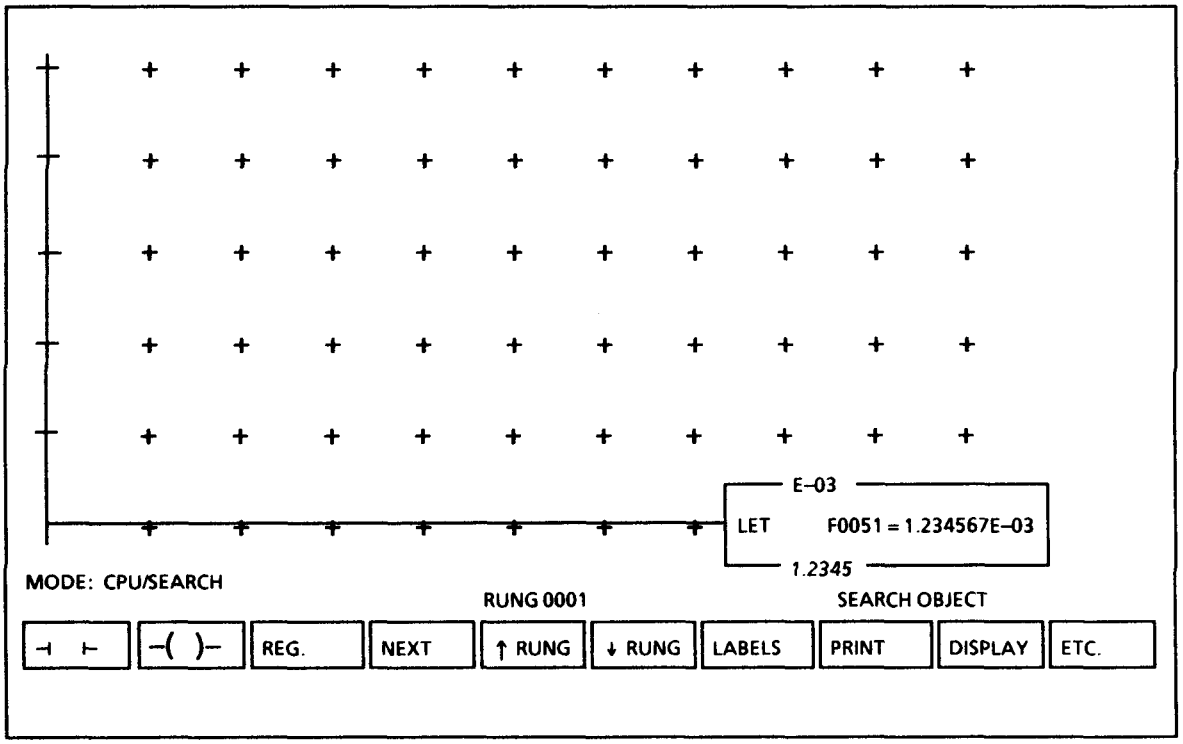


Figure 7.7 Ladder Rung Display Mode

7.8 Floating-Point Operations Within LET and IF Instructions

The Model 400 processor allows the following math functions to be programmed within "LET" and "IF" instructions.:

OPERATION	SYMBOL ON CRT
Addition	+
Subtraction	−
Multiplication	×
Division	÷
Square Root	SQRT
Sine	SIN
Cosine	COS
Common (Base 10) Logarithm	LOG
Natural (Base e) Logarithm	LN
Absolute Value	ABS
Y to the X power	Y**X

The above functions can be performed on any data values; floating point, integer, or on any constant *floating point or integer.*

SEQUENCE OF MATH OPERATIONS

Math operations are performed left to right within *LET and IF instruction boxes and may be placed* only to the right of the equal (=) or compare (<, ≥, etc.) symbol, as shown in Figure 7.8 (math operations can be placed anywhere within the dotted box).

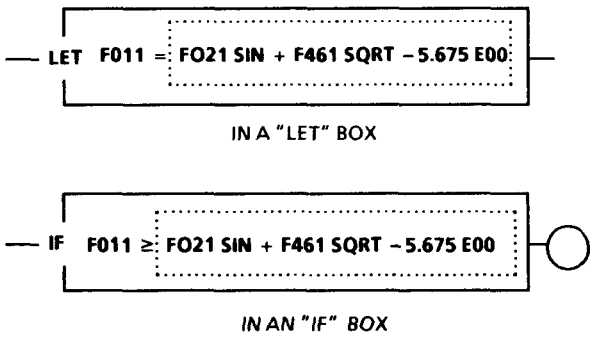


Figure 7.8 Math Operations in LET and IF Boxes

The number of math operations that can be programmed into a "LET" or "IF" rung is limited by *the areas available in one row of the programming* device's ladder matrix display, as shown in Figure 7.9.

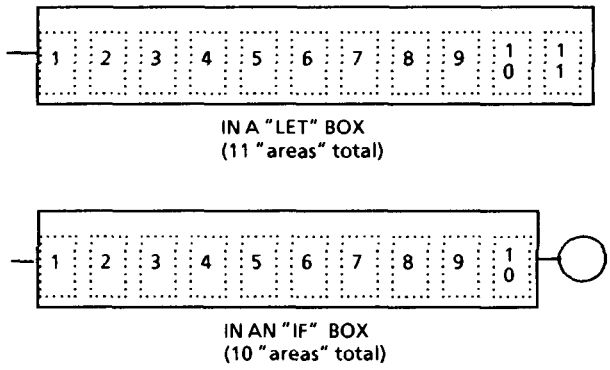


Figure 7.9 Size Limits to LET and IF Boxes

MATH PROGRAMMING CONSIDERATIONS

- The maximum number of areas available in an IF instruction is 10 when contacts are not programmed in the same rung.
- The maximum number of areas available in a *LET instruction is 11 when contacts are not* programmed in the same rung.

NOTE: *Multiple LET or IF rungs can be used if an instruction requires too many operations to be contained within one rung.*

7.9 Floating-Point Operation Using Matrixes

In many floating-point applications, the use of matrixes, in addition to single floating-point registers, will minimize memory requirements.

A floating-point matrix or array is a group of consecutively numbered floating-point registers. The rules governing the use of matrixes are explained in the Instruction Bulletin of the programming device.

7.10 Combining Floating Point With Integer Operations

Combining floating-point and integer registers in the same LET or IF instruction is allowed in the Model 400 processor. However, the user (i.e., programmer) should be very familiar with programming techniques before attempting this type of programming option.

The following sections provide examples and considerations for combining floating point and integer math within the same LET or IF instruction.

7.10.1 LET INSTRUCTIONS

If a single LET instruction contains operations performed on both integer and floating-point values, the final result is always in the floating-point format. Refer to Figure 7.10.

The final operation stores the math result into the register specified on the left of the equal sign. If the math result (floating point or integer) to the right of the equal sign does not match the register type to the left of the equal sign, the processor converts the math result to match it. The following describes four cases in which this occurs.

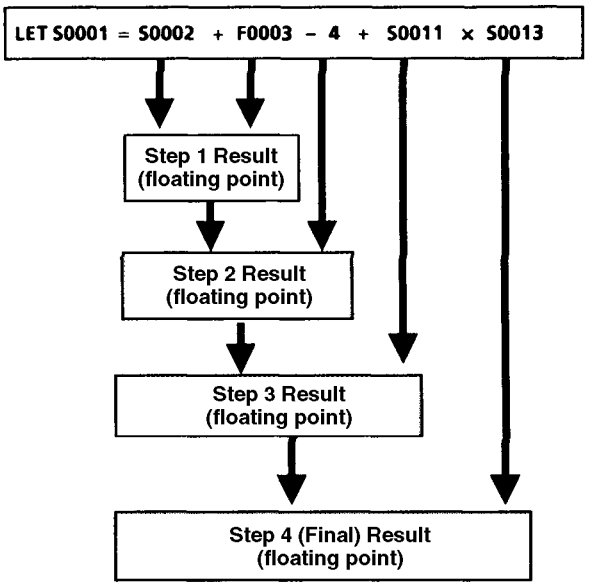
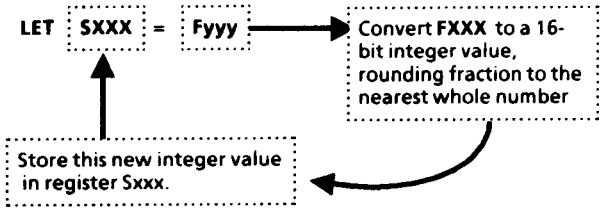


Figure 7.10 Operation Sequence in a LET Instruction Containing Both Integer and Floating-Point Registers

NOTE: *Once a floating-point value is encountered, the remaining operations are automatically be converted to floating point by the processor.*

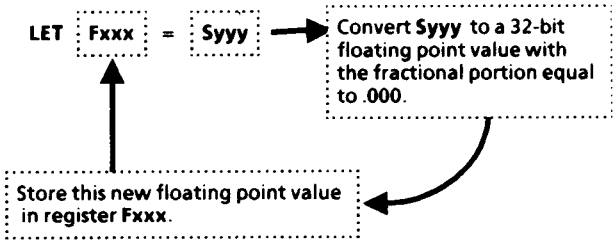
Since the processor cannot store a floating-point value in a 16-bit integer register, the floating-point value is converted to a 16-bit integer value by rounding the decimal fraction to the nearest whole number. Values less than 0.5 are rounded down, values greater than 0.5 are rounded up. For a decimal exactly equal to 0.5, the rounded result is the nearest **even** integer (for example, 2.5 is rounded down to 2.0 while 3.5 is rounded *up* to 4.0).

Case 1 - Integer Register on the Left, Floating-Point Register on the Right:

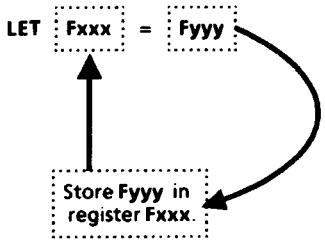


The possibility exists in Case 1 for an overflow condition. This is due to the transfer of a converted floating-point value to an integer register. Refer to Section 7.11, "Overflow Errors".

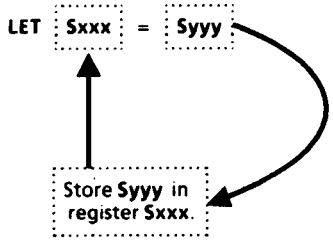
Case 2 - Floating-Point Register on Left, Integer on Right:



Case 3 - Floating-Point Register on Left, Floating-Point Result on Right:



Case 4 - Integer Register on Left, Integer Register on Right:



7.10.2 IF INSTRUCTIONS

When an IF instruction performs math operations on both integer and floating-point values, the final result on the right side of the compare will be a floating-point value. See Figure 7.11 below:

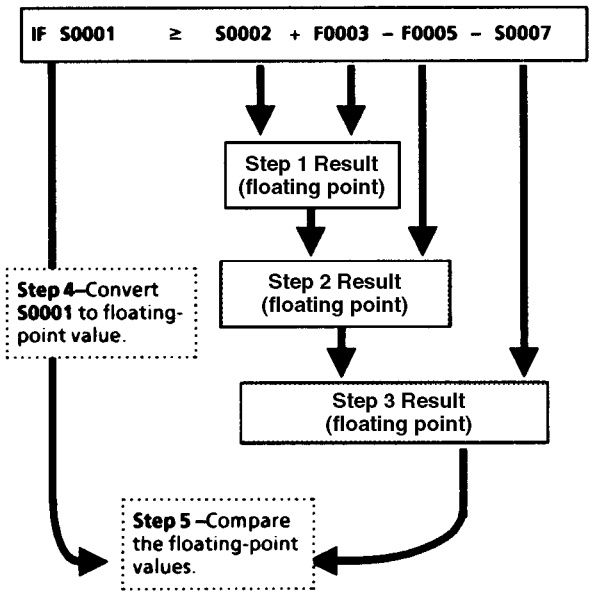
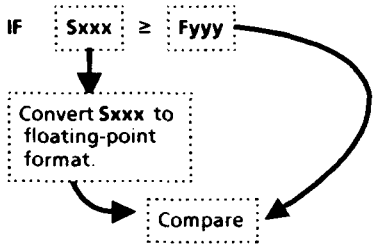


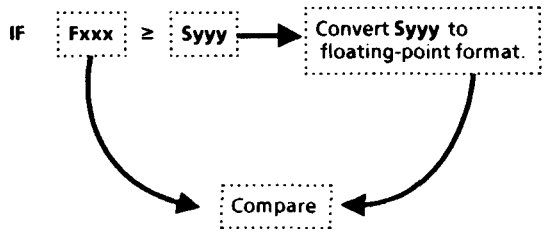
Figure 7.11 Operation Sequence in a IF Instruction That Contains Both Integer and Floating-Point Registers

If the final result of several operations is a floating-point value for comparison against an integer value, the integer value is first converted by the Model 400 to an equivalent floating-point value. The four possible "conversion/comparisons" are as follows:

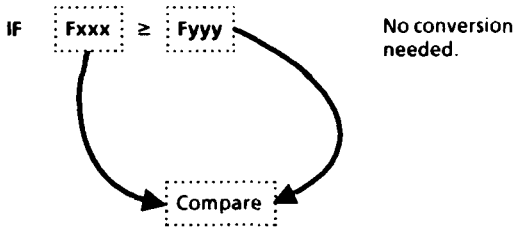
Case 1 - Compare an Integer Value to a Floating-Point Result:



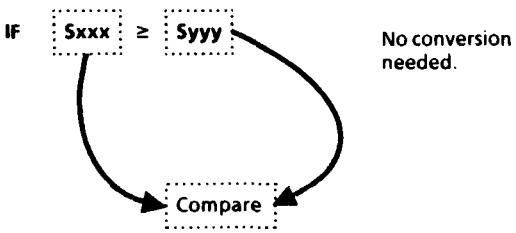
Case 2: Compare a Floating-Point Result to an Integer Value:



Case 3: Compare a Floating-Point Result to a Floating-Point Value:



Case 4: Compare an Integer Value to an Integer Result:

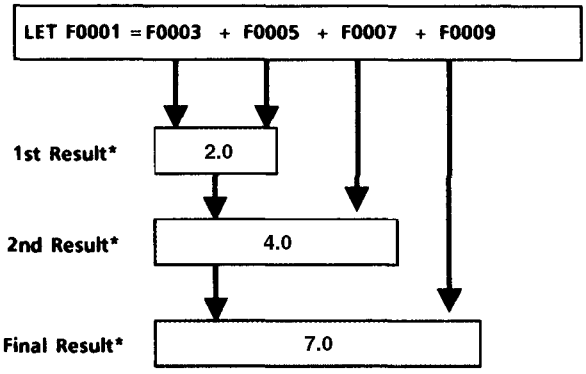


7.11 Overflow Errors

Overflow errors result from attempts to exceed a storage register's capacity.

7.11.1 *ACCUMULATOR OPERATION*

When LET instructions (data transfers) or IF instructions (data compares) containing math operations are scanned, the intermediate and final math results are temporarily stored in an *accumulator*. The accumulator contains the current running total of the math operation and is shown as a shaded rectangle in Figure 7.12 below:



*Given that F0003 = 1.0, F0005 = 1.0, F0007 = 2.0, F0009 = 3.0

Figure 7.12 Accumulator Operation

The accumulator can store a much larger value than *floating point or integer storage registers. Numbers* in the accumulator can range from $\pm 3.4 \times 10^{4932}$ to $\pm 1.2 \times 10^{-4932}$.

NOTE: *The operator has no direct control over the operation or contents of the accumulator unless one of the "Alternate Accumulator Manipulations" functions is used. Refer to Section 8.5.*

7.11.2 WHY DO OVERFLOWS OCCUR?

Overflow errors are generated any time an attempt is made to exceed a storage register's capacity. The following table shows register types and allowable storage ranges. Figure 7.14 illustrates where overflows can occur.

REGISTER TYPE	CAPACITY
Integer	± 32,767*
Integer Accumulator	± 2,147,483,647*
Floating Point	$\pm 1.2 \times 10^{-38}$ to $\pm 3.4 \times 10^{38}$
Floating-Point Accumulator	$\pm 1.2 \times 10^{-4932}$ to $\pm 3.4 \times 10^{4932}$

* *Using -32,768 or -2,147,483,648 in a math statement may produce an overflow error.*

Although the Model 400's control processor is designed to prevent the floating-point math coprocessor from generating a result of infinity from a calculation (caught by the overflow checking), infinity may be introduced into a floating-point register by loading the values shown in Figure 7.13.

REGISTER	Indication of Exceeding Positive Overrange Limit ($+3.4 \times 10^{38}$)	
	HEX	BIN/FLP
F0051	0000	None
F0052	7F80	+ INFF

REGISTER	Indication of Exceeding Negative Overrange Limit (-3.4×10^{38})	
	HEX	BIN/FLP
F0051	0000	None
F0052	FF80	-INFF

Figure 7.13 Overflow Indicators

CAUTION

Avoid entering any infinity-producing value into a floating-point register. Infinity is a mathematically invalid result that defeats the purpose of the overflow bits, causing these bits to be unreliable. Once present, infinity may propagate throughout other calculations and produce unwanted results.

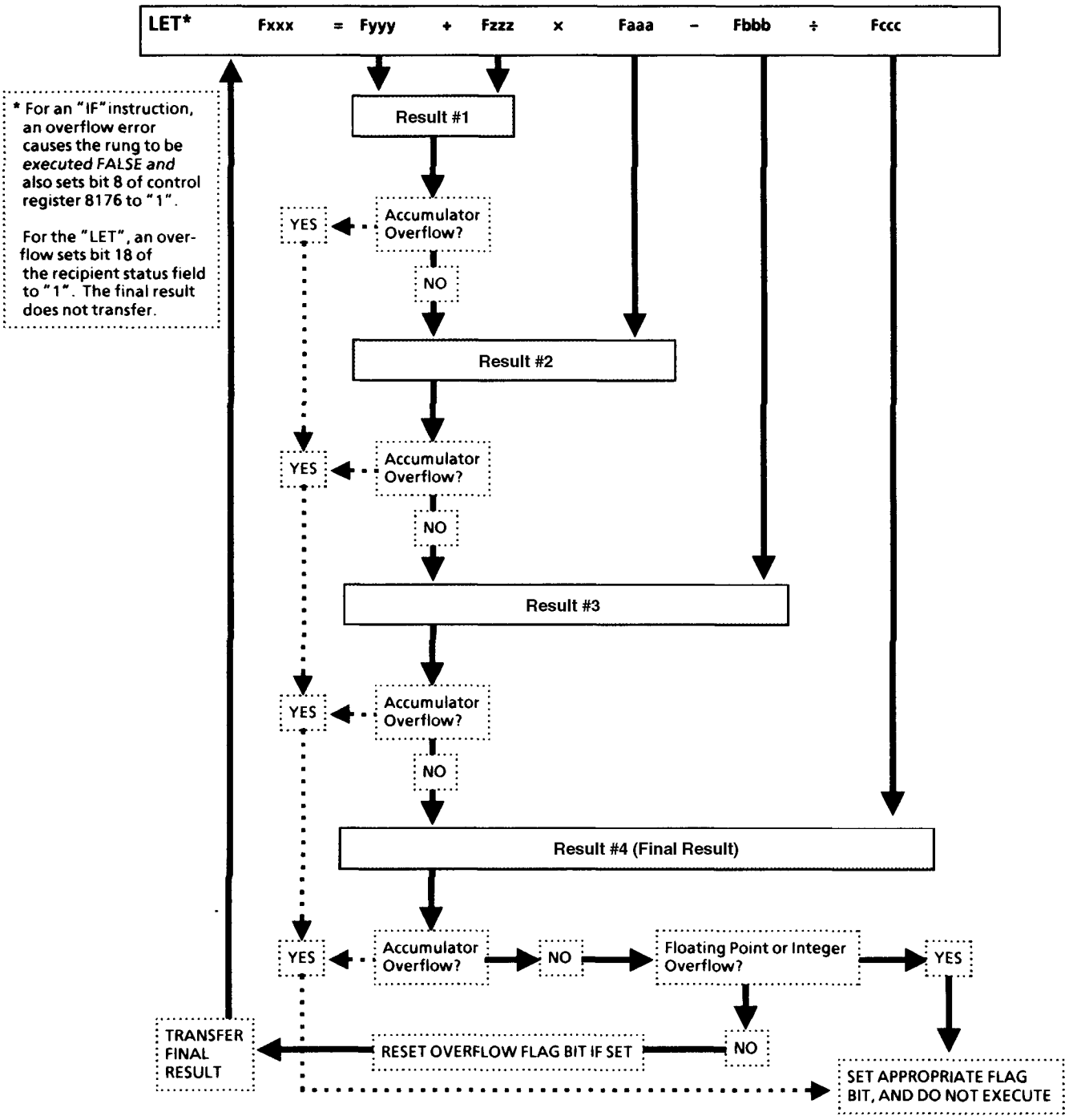


Figure 7.14 Overflow Error Points

7.12 Special Math Functions

SY/MAX programming devices such as the SPR-250, SPR-300, and SYM Programming Software use the symbols shown in the following table to represent some of the special math functions.

CRT SYMBOL	MATH FUNCTION
SIN	Sine
COS	Cosine
SQRT	Square Root
LOG	Common (base 10) Logarithm
LN	Natural (base e) Logarithm
Y**X	Raise Y to the power of X
ABS	Absolute Value

The special function always operates on the intermediate math result created by any math expressions preceding it. See Figure 7.15 below.

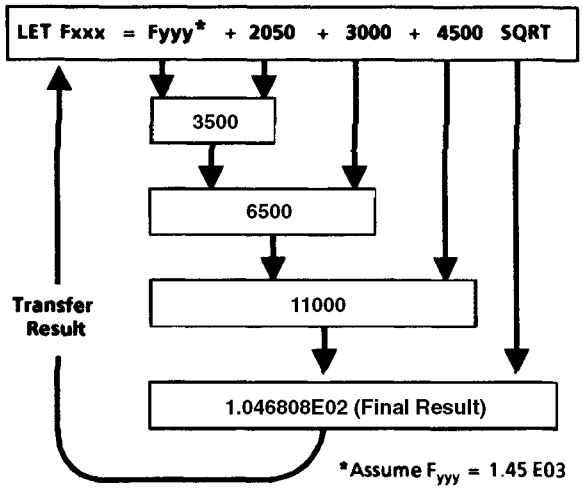


Figure 7.15 Special Function Operation

RESTRICTIONS

The value that a math function operates on is called the *argument*. Certain restrictions apply regarding the sign and magnitude of the argument.

Figure 7.16 illustrates the allowable argument conditions for some of the special math functions.

Special Function	ARGUMENT CONDITION ("Y" indicates argument *is* allowed, "N" means argument *not* allowed)					
	Zero	Final Result Over FP Range	Value −	Value +	Inter-med. Result Over FP Range	÷ By 0
SQRT in LET or IF	Y	N	N	Y	Y	N
SIN/COS in LET or IF	Y	N	Y	Y	Y	N
LOG/LN in LET or IF	N	N	Y	Y	Y	N
Y**X (Y) in LET or IF	Y	N	N	Y	Y	N
Y**X (X) in LET or IF	Y	N	Y	Y	Y	N
ABS in LET or IF	Y	N	Y	Y	Y	N

Figure 7.16 Allowable Special Function Arguments

NOTE: *Due to the behavior of the math coprocessor, small inaccuracies may be observed at function range extremes and points of discontinuity. See the following.*

TRIGONOMETRIC FUNCTION DISCONTINUITIES

The following results may be obtained when nearing points of discontinuity in calculations:

- The sine of 0°, 180°, and 360° is 0; the Model 400, however, may yield very small non-zero results.
- The tangent of 90° is infinity; the Model 400 may report it to be an extremely *large* number (–3.7E19, typically).
- The tangent of 180° is 0; the Model 400 may show it to be an extremely *small* number.

Results similar to the above could occur when using any trig function. If a problem is anticipated, it should be compensated for in the user program.

8 SPECIAL INSTRUCTIONS

8.1 Description

The Model 400 instruction set has been expanded beyond the use of floating-point five-function math, sine, cosine, logarithms (base 10 and natural), absolute value, and exponential math. Refer to Section 7 for a discussion of floating-point math and the utilization of the listed functions.

NOTE: *It is strongly recommended that the concepts in Section 7 be understood before proceeding with this section.*

Programming the following instructions is accomplished by pressing the SPECIAL soft key, accessible when constructing an IF or LET rung, followed by the corresponding function number associated with the instruction. The appropriate function is performed on the intermediate mathematical result that exists just prior to encountering the SPECIAL instruction.

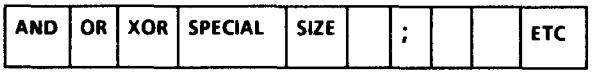
NOTE: *All mathematical operations are performed from left to right and each operation is completed before the next operation in the rung is executed.*

Certain functions, by definition, require a *double argument.* The user should program a semicolon (from the soft key display) after the function number, followed by the second argument (which may be either a constant or a register value).

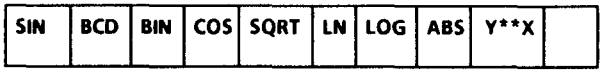
Programming any of the single argument functions results in the indicated operation being performed on the intermediate result. To program, the user must begin constructing a rung using the IF or LET box format. The desired operation can then be applied to any intermediate result which exists to the right of the "=" sign.

To implement the desired function, begin with the following "starting point" soft key display:

Striking the ETC softkey produces a new display which looks like the following:



Striking the SPECIAL soft key produces the following display:



The user may now strike the proper soft key if one of the listed functions is to be used, or may instead enter from the keyboard the numeric value which corresponds to one of the additional SPECIAL functions. In the latter case, the entered number appears in the IF or LET rung box and the soft key display reverts to the "starting point" soft key display.

After entering the desired numeric value, the user may either exit the box (if finished constructing the mathematical expression) or continue building the expression by following the SPECIAL instruction with either another SPECIAL instruction or a mathematical operator (+, -, X, ÷). In either case, the abbreviation SPEC appears in the box to indicate the appropriate SPECIAL function has been entered in the rung.

Programming the double argument functions is similar except that the user has to provide the *second argument in order for the function to be* programmed correctly. This must be provided IMMEDIATELY after the numeric value which identifies the SPECIAL double argument function. Thus, the keystrokes employed for the double argument functions are the same as for the single argument function, but require the following additional keystrokes:

1. After entering the numeric value which corresponds to the desired SPECIAL function, the soft key display reverts to the "starting point" display:

REG	MATRIX	FLP REG	FLP MAT	FLP IMD	+	-	X	÷	ETC

2. *Striking the ETC soft key changes the display to* the following:

AND	OR	XOR	SPECIAL	SIZE		;			ETC

3. Striking the semicolon (;) soft key, followed by the ETC key (the display does not change after striking the ";" soft key) prepares the box for the second argument. The following is displayed:

REG	MATRIX	FLP REG	FLP MAT	FLP IMD	+	-	X	÷	ETC

The second argument can now be entered as either a constant (integer or floating point number) or a variable (integer register or floating point register). Upon entering the second argument, the user may either exit the box (if finished constructing the mathematical expression) or continue building the expression by entering either another SPECIAL instruction or a mathematical operator (+, -, X, ÷). In either case, the abbreviation "SPEC" appears in the box to indicate the appropriate SPECIAL instruction has been entered in the rung.

PROGRAMMING CONSIDERATIONS

- The first element to the right of the "=" sign cannot be a mathematical operator (+,-, X, or ÷) *or a SPECIAL function.*
- Entering a numeric value for a SPECIAL function that does not exist (i.e., a "Special Number" greater than 65) generates PROCESSOR ERROR 5 when the rung is loaded.
- When programming SPECIAL functions with two arguments, the *EXACT* order of entry is critical. The numeric value for the function *must be followed immediately by the semicolon from the softkey display, which in turn must be followed immediately by the second argument.*

Along with the terminology in Section 7.2, familiarity with the following terms is necessary to understand the rest of this section:

Accumulator ..... An 80-bit "scratch-pad" register which contains the intermediate result of the Model 400's floating point calculations. Also discussed in Section 7.11.

Alternate Accumulator A different 80-bit register that can be accessed by the user to provide *precision that would* normally be lost in using a normal 32-bit floating-point register.

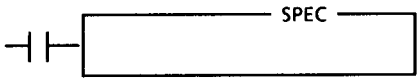
Intermediate Result (IR) The data register immediately to the right of the "=" sign, or the current running total of a computation. The IR is stored in the accumulator.

New Intermediate Result (NIR) ..... The result obtained after applying a SPECIAL function to the intermediate result.

ARG2 ..... The notation for the second argument of a double-argument function.

ALT ..... Notation for the contents of the alternate accumulator.

Figure 8.1 lists all the special (SPEC) instructions the Model 400 processor can perform. The SPEC portion of the rung is shown below:



Model 400 Types SCP-423, 424 and 444 are capable of performing ALL the listed instructions. The Type SCP-401 can only perform the instructions that are marked with an asterisk (*).

The sample rungs used in Figure 8.1 are for illustrative purposes only and are not intended to produce practical results. Most of the registers shown are integer type registers. For certain functions (in particular the trig functions), a *floating point* register would produce more accurate results (in Types SCP-423, 424, and 444 only). Also, IF statements could have been used in place of the LETs in most cases.

SPEC #	Function	Action	Example
0	SIN(X)	Calculate sine of x	LET F11 = F21 SPEC SIN
1*	BIN to BCD	Convert binary pattern to Binary Coded Decimal	LET S10 = S20 SPEC BCD
2*	BCD to BIN	Convert BCD value to binary	LET S10 = S20 SPEC BIN
3	COS(X)	Calculate cosine of x	LET F11 = F21 SPEC COS
4*	SQRT	Calculate square root of a positive value	LET S10 = S20 SPEC SQR
5	LN(X)	Calculate natural log of x	LET F11 = F21 SPEC LN
6	LOG(X)	Calculate common log of x	LET F11 = F21 SPEC LOG
7*	|X|	Calculate absolute value of x	LET S10 = S20 SPEC ABS
8	Y**X	Raise Y to the power of X	LET S10 = S20 Y**X; S30 SPEC
9	Y**ALT	Raise Y to the power of the value stored in the alternate accumulator	LET S10 = S20 SPEC 9
10	1/X	Calculate the reciprocal of x	LET F11 = F21 SPEC 10
11*	SWAP 16	High byte and low byte of IR are exchanged	LET S10 = S20 SPEC 11
12*	SWAP 32	Most significant word in the IR becomes the NIR	LET S10 = S20 SPEC 12
13*	READ STAT	Indirect register read of status field (data in S20 points to register where status field is found)	LET S10 = S20 SPEC 13
14*	READ DATA	Indirect register read of data field (data in S20 points to the register where data field is located)	LET S10 = S20 SPEC 14
15*	FIND BIT	Indirect register bit search (find and reset the lowest bit set in S20. Store 1 to 16 to identify bit, or store –32,768 if all bits are at "0")	LET S10 = S20 SPEC 15
16*	NEGATE	Change sign of current number	LET S10 = S20 SPEC 16
17*	NUM TYPE	Returns type of current number	LET S10 = S20 SPEC 17
18*	ACC ALT	Exchange current and alternate accumulators	LET S10 = S20 SPEC 18
19*	ADD ALT	Add S20 to alternate accumulator	LET S10 = S20 SPEC 19
20*	SUB ALT	Subtract S20 from alternate accumulator	LET S10 = S20 SPEC 20
21*	DUP ALT	Copy alternate accumulator into current accumulator	LET S10 = S20 SPEC 21
22*	DUP ACC	Copy current accumulator into alternate accumulator	LET S10 = S20 SPEC 22
23*	RAN-DOM	Generates a random number (from 0 to value of current accumulator minus 1)	LET S10 = S20 SPEC 23
24	ROUND	Rounds FP numbers to integers	LET S10 = F21 SPEC 24
25	SQUARE	Squares current result	LET S10 = S20 SPEC 25
26	HYPOT	Calculate two-dimensional hypotenuse $\sqrt{IR^2 + ARG2^2}$	LET F11 = F21 SPEC 26; F31
27	HYPALT	Calculate two-dimensional hypotenuse $\sqrt{IR^2 + ALT^2}$	LET F11 = F21 SPEC 27

Figure 8.1 Special Instruction List

SPEC #	Function	Action	Example
28	HYP XYZ	Calculate three-dimensional hypotenuse $\sqrt{IR^2 + ARG2^2 + ALT^2}$	LET F11 = F21 SPEC 28;F31
29	ADD STAT	Perform statistical summation S20 = IR = data, S30 = ARG2 = pointer	LET S10 = S20 SPEC 29;S30
30	VAR-IANCE	Calculate variance on statistical summation	LET F11 = S30 SPEC 30
31	TAN(X)	Calculate tangent of x	LET F11 = F21 SPEC 31
32	COSEC (X)	Calculate 1/sin(x)	LET F11 = F21 SPEC 32
33	SECANT (X)	Calculate 1/cos(x)	LET F11 = F21 SPEC 33
34	COTAN (X)	Calculate 1/tan(x)	LET F11 = F21 SPEC 34
35	ARCSIN (X)	Calculate $\sin^{-1}(x)$	LET F11 = F21 SPEC 35
36	ARCCOS (X)	Calculate $\cos^{-1}(x)$	LET F11 = F21 SPEC 36
37	ARCTAN (X)	Calculate $\tan^{-1}(x)$	LET F11 = F21 SPEC 37
38	ARCTAN (Y/X)	Calculate $\tan^{-1}(y/x)$	LET F11 = F21 SPEC 38;F31
39	ARCCSC (X)	Calculate $\sin^{-1}(1/x)$	LET F11 = F21 SPEC 39
40	ARCSEC (X)	Calculate $\cos^{-1}(1/x)$	LET F11 = F21 SPEC 40
41	ARCCOT (X)	Calculate $\tan^{-1}(1/x)$	LET F11 = F21 SPEC 41
42	SET DEGREE	Set degree mode for trig calculations	LET F11 = F21 SPEC 42
43	SET RADIAN	Set radian mode for trig calculations	LET F11 = F21 SPEC 43
44	DEG RAD	Convert degrees to radians	LET F11 = F21 SPEC 44
45	RAD DEG	Convert radians to degrees	LET F11 = F21 SPEC 45
46	PID REV	Calculate reverse-acting PID loop	LET S517 = 51 SPEC 46
47	PID DIR	Calculate direct-acting PID loop	LET S517 = 51 SPEC 47
48	PID MAN	Calculate manual mode for PID loop	LET S517 = 51 SPEC 48
49	LEAD /LAG	Perform the lead/lag function on a process variable	LET S56 = 71 SPEC 49
50	PI	Multiply by pi (3.1415....)	LET F11 = F21 SPEC 50
51	L2T	Multiply by log base 2 of 10	LET F11 = F21 SPEC 51
52	L2E	Multiply by log base 2 of e	LET F11 = F21 SPEC 52
53	LG2	Multiply by log base 10 of 2	LET F11 = F21 SPEC 53
54	LN2	Multiply by log base e of 2	LET F11 = F21 SPEC 54
55	EULER	Multiply by Euler's Constant	LET F11 = F21 SPEC 55
56	E	Multiply by e (natural logarithm)	LET F11 = F21 SPEC 56
57	1 DEG	Multiply by one degree (in radians)	LET F11 = F21 SPEC 57
58	1 RAD	Multiply by one radian (in degrees)	LET F11 = F21 SPEC 58
59	E**PI	Multiply by e raised to the power of pi	LET F11 = F21 SPEC 59

Figure 8.1 Special Instruction List (continued)

8.2 Math and Trig Operations

The operations listed in Figure 8.2 are designed to enhance the functions listed in the programmer's Instruction Bulletin. A number of high-precision constants are available in the Model 400, along with the ability to perform trig operations in either degree or radian mode.

NOTE: ***Any function that requires floating-point math is not available on the Type SCP-401 processor.***

APPLICATION CONSIDERATIONS

- The default mode for trig operations is *degrees* (initially set upon a HALT-to-RUN transition).
- The selected mode remains in effect until either a SPEC 42 or SPEC 43 is encountered or until a HALT-to-RUN transition occurs.
- Although the CRT is limited to displaying seven figures, the internal accumulator precision remains at 80 bits and the floating-point register's precision is 32 bits.
- All other rules for IF and LET functions apply.

In Figure 8.2 the second column shows the "Special Number" which is used in the LET box to generate that function. The SPEC number is the same that appears in the first column of the preceding table.

Instruction	SPEC #	Description
INVERT	10	Takes the reciprocal of the intermediate result (NIR = 1/IR)
NEGATE	16	Changes the sign of the intermediate result (NIR = 0–IR)
ROUND	24	Rounds the current intermediate result to the nearest integer number. The value 0.5 is converted to the nearest *even* integer.
XSQRD	25	*Squares the intermediate* result
HYPOT	26	Calculates the two-dimensional hypotenuse. This is a *double-argument* function.
HYPXYZ	28	Calculates the three-dimensional hypotenuse, a *double-argument* function.
SETDEG	42	All subsequent calculations will be performed in *degrees*
SETRAD	43	All subsequent calculations will be performed in *radians*
DEGRAD	44	Converts degrees to radians
RADDEG	45	Converts radians to degrees
TANGNT	31	Calculates the tangent
COSEC	32	Calculates the cosecant
SECANT	33	Calculates the secant
COTAN	34	Calculates the cotangent
ARCSIN	35	Calculates the arc sine
ARCCOS	36	Calculates the arc cosine
ARCTAN	37	Calculates the arc tangent
ARCCSC	39	Calculates the arc cosecant
ARCSEC	40	Calculates the arc secant
ARCCOT	41	Calculates the arc cotangent

Figure 8.2 Special Trig/Math Operations

TRIG AND MATH CONSTANTS

Figure 8.3 below gives a list of constants available for use in the Model 400 programming set. Entering the numeric function results in the intermediate result being multiplied by the indicated constant, which is stored to 80-bit (18 significant digit) precision.

Instruction	SPEC #	IR Will be Multiplied by:
PI	50	Pi to 18 significant digits
L2T	51	Log base 2 of 10
L2E	52	Log base 2 of the value of e
LG2	53	Log base 10 of 2
LN2	54	Log base e of 2
EULER	55	Euler's Constant
E	56	Natural log e
1DEG	57	One degree, in radians
1RAD	58	One radian, in degrees
ETOPI	59	e raised to the power of *pi*
PITOE	60	*pi* raised to the power of e
ETOE	*61*	*e raised to the power of e*
ETOEULR	62	e raised to the power of Euler's Constant
SQRTE	63	The square root of e
SQRTPI	64	The square root of *pi*

Figure 8.3 Special Constants

8.3 Statistical Operations

The number preceding SPECIAL 29 in the LET box (X) represents the numeric quantity to be operated on and then stored in the statistical block. Refer to Figure 8.4. The number can be contained in either an integer or floating-point register.

The second argument (which follows the semicolon) may be a constant or a variable and must be an *odd* integer ranging from 1 to 3995. This integer is used as a pointer to a user-specified block of 5 registers of accumulated data. Note that the first four registers are actually a pair of floating-point registers and the fifth is an integer register.

NOTE: *The new intermediate result in the following table is denoted as "N".*

Instruction	SPEC #	Statistical Operation
ADDSTA	29	Adds the value preceding the SPEC 29 entry to the statistical block of registers (defined by the pointer), as follows: **F1 = SUM(X) = ΣX** **F3 = SUM(X^2) = ΣX^2** **S5 = N, where N is the number of terms which have been summed.** To be of practical value, the LET statement that initiates the ADDSTA function must be *transition-sensitive*. If it isn't, the summation operations will be performed every scan. The maximum legal value for N is +32,767.
VARNCE	30	Calculate the variance on the accumulated statistical block. The number preceding the SPEC 30 in the LET box must point to the same block of registers used by the ADDSTA function. $NIR = \frac{N*[SUM(X^2)] - [SUM(X)]^2}{N*(N-1)}$ $= \frac{N * \Sigma X^2 - (\Sigma X)^2}{N*(N-1)}$ Since the *standard deviation* is, by definition, the square root of the variance, the standard deviation can also be obtained by taking the square root of the result of the VARNCE operation. In the same way, the *mean* can be obtained by dividing the first floating point register assigned to the statistical block by the fifth, as in the following: $MEAN = \frac{SUM(X)}{N} = \frac{\Sigma X}{N} = \frac{F1}{S5}$

Figure 8.4 Statistical Functions

8.4 Indirect Register Read Operations

For the functions listed in Figure 8.5, the number preceding the SPEC function in the LET box acts as an indirect pointer to the desired register. The indirect pointer must be an integer (representing a number or a storage register) ranging from 1 to 4000 or 8097 to 8192.

Instruction	SPEC #	Read Operation
RDSTAT	13	Reads the *status* field of the indicated indirect register. The status field becomes the new IR (NIR = status field of the register pointed at by the IR).
RDDATA	14	Reads the *data* field of the indicated indirect register. The data field becomes the new IR (NIR = data field of the register pointed at by the IR).
FNDBIT	15	Locates and identifies the lowest bit set in the indirect register data field. The lowest bit position that is set becomes the new IR, then the bit itself is reset. If no bits are found to be set, the NIR becomes 8000H (-32,768). Thus, the only values which the NIR can become after execution of FNDBIT are 1 to 16 (if bit IS found), or -32,768 (if bit ISN'T found). To be of practical value, the LET statement that initiates the FNDBIT function should be *transition sensitive*. If not, the operation is performed every scan and the user may not perceive proper operation.

Figure 8.5 Register READ Operations

Instruction	SPEC #	Manipulation Performed
YTOALT	9	Similar to the math operation Y^x, except that Y is raised to the power of the alternate accumulator (NIR = IR^{ALT})
ACCALT	18	Exchanges the intermediate result with the alternate accumulator (IR becomes ALT and ALT becomes IR)
ADDALT	19	Adds the intermediate result to the alternate accumulator. The intermediate result remains the same.
SUBALT	20	Subtracts the intermediate result from the alternate accumulator. The intermediate result remains the same.
DUPALT	21	Duplicates the alternate accumulator. The NIR assumes the value in the alternate accumulator (NIR = ALT), and the alternate accumulator remains unchanged.
DUPACC	22	Duplicates the accumulator (intermediate result). The intermediate result is copied into the alternate accumulator (ALT = NIR), and the IR remains unchanged.
HYPALT	27	Calculates the hypotenuse based on results stored in the alternate and IR accumulators [NIR = $(\sqrt{IR^2 + ALT^2})$
ATANYX	38	Calculates the arc tangent for the ratio of the intermediate result to the second argument [NIR = ARCTAN (IR/ARG2)]

Figure 8.6 Accumulator Manipulations

8.5 Alternate Accumulator Manipulations

The functions in Figure 8.6 allow access to the two 80-bit floating-point accumulators in the Model 400 Type SCP-423, 424, and 444. These accumulators maintain a high accuracy which might otherwise be lost in a standard 32-bit floating point register.

8.6 Bus Write to Microcell

Instruction	SPEC #	Action Performed
BWRITE	67	Performs an immediate write of data to the specified register, with the count value indicated by the second argument.

Figure 8.7 Bus Write to Microcell

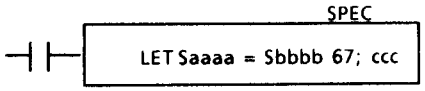
When exchanging data with the Microcell via the backplane, the image table operation of the Model 400 processor requires some special considerations not explicitly stated in Section 8 or Section 11.4 of the Microcell Instruction Bulletin #30598-275-xx. The following technique for reading and writing registers applies to **Model 400 processors, Revision 3.6 or later**, used with Microcells which are Revision 4.0 and later. Earlier revision Microcells will allow Model 400 processors to write data over the backplane, but not to read any registers.

Reading Registers

All rack addressed Microcell registers (maximum of 256) are read by the processor into its image table as part of normal End-of-Scan updating.

Writing Registers

Any rack addressed Microcell registers can be written to the processor by utilizing the following SPECIAL command.



where: aaaa = first Microcell destination register.
bbbb = processor source register
ccc = register count

SPECIAL 67 is similar to an Immediate I/O Update instruction and results in the contents of the block of processor registers starting at address bbbb to be written to the block of Microcell registers starting at address aaaa, with the block size determined by the register count ccc. Note this is different from a standard LET statement which only transfers data within the image table registers of the processor. The following application considerations apply:

- The processor must be Rev. 3.6 or greater; the Microcell must be Rev. 4.0 or greater.
- The format must be exactly as shown above; bit 18 in the status field of reg. aaaa will be set to indicate the instruction has failed to execute due to improper formatting.
- The Microcell must be located in the same rack as the processor.
- A minimum of 28 registers (maximum of 256) should be rack addressed to the Microcell.
- The SPECIAL 67 instruction, Figure 8.7, is intended to be used only for performing direct bus updates of Microcell registers, and should not be applied to other modules (use Immediate I/O Update instruction instead).

The following examples illustrate the use of the SPECIAL WRITE command to update registers in the Microcell. Note any of these registers can be READ using standard (e.g. IF) instructions.

Example 1 – Updating the FILE COMMAND Registers

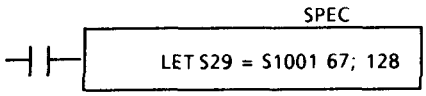
Section 11.4 of the Microcell Instruction Bulletin #30598-275-xx shows three examples for downloading a file, appending an MCM file, and shutting down the MCM by writing to the first 28 Microcell registers. Each example uses a ladder rung employing a MATRIX statement to accomplish the register data transfers. Each of these statements needs to be replaced by the SPECIAL 67 format, with the "PNTR" value being replaced by the "count" value of the new instruction.

Thus, the rung of section 11.4.1 now becomes

Example 2 – Reading and Writing the BUS DATA Registers

Assume the first 156 system registers have been rack addressed to the Microcell. Registers 29-156 are the BUS DATA registers, available for reading and writing register values. Because these Microcell registers are automatically read into the processor image table at the end of each scan, they can be monitored with the standard instruction set.

In order to write the contents of (for example) registers 1001-1128 into these registers, use the SPECIAL instruction.



Note if a regular LET or MATRIX statement was used instead to update registers 29 through 156, only the processor's *internal image table registers would* be updated; these registers would subsequently be overwritten when the corresponding Microcell registers were automatically read during normal End-of-Scan updating.

8.7 Miscellaneous

The table in Figure 8.8 lists the various miscellaneous instructions that are available with the Model 400 processor.

Instruction	SPEC #	Action Performed
RANDOM	23	Generates a random number. The number preceding the SPEC 23 box is used as the argument value for the generator. *The new IR will fall* in the range 0 to (IR–1). **NOTE:** *An argument value of 0 or 1 will return a random number value of 0, while an argument greater than 32,767 or less than 0 produces an error.*
SWAP 16	11	Exchanges the high byte (9-16) and low byte (1-8) of the IR. The IR must be an integer in the range ± 32,767.
SWAP 32	12	Copies the high word of the IR to the low word of the IR. The high word is either all "0"s (if bit 16 is a "0"), or all "1"s (if bit 16 is a "1").
NUMTYP	17	Identifies the number type of the intermediate result. The IR becomes a value from 0 to 14, according to standards applied to the Motorola 68010 microprocessor and 68881 math coprocessor as shown below: 0 Positive normalized or de-normalized 1 Positive not-a-number 2 Positive infinity 3 *Unused* 4 Positive zero 5-7 *Unused* 8 Negative normalized or de-normalized 9 Negative not-a-number 10 Negative infinity 11 Integer zero 12 Negative zero 13 Negative integer 14 Positive integer

Figure 8.8 Miscellaneous Operations

9 PID OPERATIONS
(Not Applicable to SCP-401)

9.1 Introduction

NOTE: *A more complete discussion of the equations and terminologies used in this section can be found in Instruction Bulletin 30598-301-0X ("PID Closed Loop Control") or 30598-240-0X ("Process Control Module").*

Algorithms are available in the Model 400 processor for solving the following three types of PID loops:

1. Reverse-acting (PIDR)
2. Direct-acting (PIDD)
3. Manual (PIDM)

The number that precedes SPEC 46, 47 or 48 in the Special Instruction LET box (an odd integer from 1 to 3983) is a pointer to a user-defined block of 18 storage registers.

9.2 Register Allocation

The block of 18 storage registers provides information needed by the Model 400 to execute the PID algorithms, and must follow the format shown in Figure 9.1.

The "L" prefix in Figure 9.1 indicates the relative position within the 18-register block. For example, if registers 51 through 68 are set aside for the PID algorithm registers, "L1" would refer to register 51, "L5" would refer to register 55, and so on. The boldface items in Figure 9.1 are parameters used in equations.

REGISTER #	*PURPOSE*
L1, L2	Integral Summation [**SUM (i)**]. These two registers must be a floating-point register.
L3	Value of the process variable used the last time the loop was updated (**PVn–1**)
L4	Not Used
L5	*The output from the loop solution* (**OUTPUT**)
L6	Value of the process variable used during the present loop update (**PVn**)
L7	The set point (**SP**)
L8	LOOP STATUS *BIT 2*: 1 if loop *is not* in manual *0 if loop is in manual* *BIT 3*: 1 if loop *is* in automatic 0 if loop *is not* in automatic *BIT 10*: 1 if loop is *direct-acting* 0 if loop is *reverse-acting* *BIT 14*: 1 if high output limit exceeded 0 if high output limit *not* exceeded *BIT 15*: 1 if low output limit exceeded 0 if low output limit *not* exceeded *BIT 16*: 1 if low output limit is illegally set to be higher than the high limit
L9, L10	Not Used
L11	Proportional (gain) constant (K_p) This is a dimensionless number.
L12	Integral (reset) constant (K_I) in repeats per minute
L13	Derivative (rate) constant (K_D) in seconds
L14	Bias or offset (**B**) in %
L15	High output limit, in %
L16	Low output limit, in %
L17	Not Used
L18	Sample interval (T_S) in milliseconds

Figure 9.1 PID Register Allocation

PROGRAMMING CONSIDERATIONS

- The tuning parameter values of **SP**, K_p, K_I, K_D, and **B** are entered with programming equipment *or in the user program.*

- Registers L5, L6, L7, L14, L15, and L16 are entered to reflect a percent of scale from 0000 to 9999. The decimal point is implied as XX.XX. *For example, an OUTPUT of 50% is entered as* 5000, a PROCESS VARIABLE value of 7475 represents 74.75% of full-scale, etc.

- Registers L11, L12, and L13 are entered in the range of 0 to 32,767 with the decimal point implied as XXX.XX. For example, a gain of 1.00 is entered as 100, a reset of 27 that repeats per minute is entered as 2700, while a rate of 150 seconds is entered as 15000.

- Register L18's (sample interval) implied decimal point is XX.XXX. For example, a sample interval of 200 milliseconds is entered as 200, while a sample interval of 10 seconds is entered as 10,000.

- The high and low output limits are entered in the same format as the tuning parameters, as a percent of full-scale.

- The loop algorithm calculates **SUM (I)** and **OUTPUT**, and stores PV_{n-1} for the next calculation. The user needs to input PV_n and transfer the calculated **OUTPUT** to a real-world analog output register. These manipulations typically occur through ladder rungs.

- Additional ladder rungs are required to set up a *total PID function. See the example in Section* 9.7.

- In order to ensure the integrity of the calculations, Rev. 3 and later revisions of the Model 400 perform limit checks on the variables loaded by the user. Valid ranges are defined as follows:

$0 \leq X \leq 9999$	for OUTPUT, PV, SP, Bias, and High and Low Output Limits.
$X \geq 0$	for K_p, K_I, and K_D.
$X > 0$	for sample interval T_s

If any of these range limits are not observed, the PID algorithm is not executed and bit 18 of the destination register (the register to the left of the "=" sign in the PID rung) is set to flag the error. In addition, the processor sets bit 18 of the out-of-range register to facilitate troubleshooting by identifying the source of the illegal value.

- In Revision 1 of the Model 400, it is necessary to rack address the block of storage registers used in the PID algorithm. To accomplish this, assign the remaining number of the 4000 processor registers to the first **unused** slot in the CPU rack. This allows the algorithm to execute properly and does not effect the scan speed, throughput, or update time.

9.3 Reverse-Acting Loop

The SPEC 46 entry in a LET box calculates a reverse-acting (PIDR) loop in the following sequence:

1. $\mathbf{SUM\ (I)} = \dfrac{K_P * K_I * T_S * (SP - PV_n)}{6 * 10^{10}} + \mathbf{SUM(I)}$

2. LET L1, L2 = **SUM (I)**

3. $\mathbf{OUTPUT} = \left[\mathbf{SUM\ (I)} + \dfrac{K_p * (SP - PV_n)}{10{,}000} + \dfrac{K_p * K_D * (PV_{n-1} - PV_n)}{T_S * 1000} + \dfrac{B}{100}\right] * 100$

4. LET $PV_{n-1} = PV_n$

5. *CLEAR* bit 10 of register 8 (reverse-acting flag)

6. If low output limit exceeds high output limit, set bit 16 of register 8 to flag this illegal condition

7. Check if output exceeds the high output limit. If YES, and $K_I \neq 0$, a back calculation is performed on the integral sum.

IF **OUTPUT** > High output limit:

A. Set bit 14 of register 8

B. LET L1 = $\dfrac{\text{high output limit} - \mathbf{OUTPUT}}{100} + \mathbf{SUM(I)}$

C. LET **OUTPUT** = high output limit

8. If **OUTPUT** < (high output limit + 1), reset bit 14 of register 8 to indicate that high limit is not exceeded.

9. Check if output is below the low output limit. If YES,and $K_I \neq 0$, a back calculation is performed on the integral sum.

IF **OUTPUT** < Low output limit:

A. Set bit 15 of register 8

B. LET L1 = $\dfrac{\textit{low output limit} - \mathbf{OUTPUT}}{100} + \mathbf{SUM(I)}$

C. LET **OUTPUT** = low output limit

10. If **OUTPUT** ≥ low output limit, reset bit 15 of register 8 to indicate that low limit is not exceeded.

The final action in the Reverse-Acting Loop is to store the newly-calculated loop **OUTPUT** value in L5 and in the SPEC 46 LET box register which is just to the left of the "=" sign.

9.4 Direct-Acting Loop

The SPEC 47 LET box calculates a direct-acting (PIDD) loop. The algorithm used is identical to the one used for the reverse-acting loop, except that the sign for the mathematical difference between **SP** and **PV** *terms is reversed.*

The change in sign results in changes to the first, third, and fifth steps of the sequence in Section 9.3. All other steps are the same as shown in Section 9.3.

As with the PIDR, the newly-calculated loop **OUTPUT** value is stored both in register L5 and in the register to the left of the "=" sign in the SPEC 47 LET box.

1. $\mathbf{SUM\ (I)} = \dfrac{K_P * K_I * T_S * (P_{Vn} - SP)}{6 * 10^{10}} + \mathbf{SUM(I)}$

2. SAME AS PIDR

3. $\text{OUTPUT} = \left[\mathbf{SUM\ (I)} + \dfrac{K_p * (PV_n - SP)}{10{,}000} + \dfrac{K_p * K_D * (PV_n - PV_{n-1})}{T_S * 1000} + \dfrac{B}{100}\right] * 100$

4. SAME AS PIDR

5. SET bit 10 of register 8 (direct-acting flag)

6. SAME AS PIDR

7. SAME AS PIDR

8. SAME AS PIDR

9. SAME AS PIDR

10. SAME AS PIDR

9.5 Manual Loop

The SPEC 48 instruction in a LET box causes the PID loop to be manually (PIDM) controlled. The number preceding SPEC 48 in the LET box must point to the same block of 18 registers as did the PIDR or PIDD loop algorithm.

The PIDM loop is executed as follows to facilitate a bumpless transfer to AUTO:

1. LET **SP** = $\mathbf{PV_n}$ (sets the setpoint equal to the process variable)
2. LET **B** = **OUTPUT** (sets bias equal to the loop output)
3. LET **SUM(I)** = 0 (zeroes out the integral summation)
4. LET $\mathbf{PV_{n-1}} = \mathbf{PV_n}$
5. The final action in the Manual-Acting loop is to transfer the OUTPUT Value in L5 to the SPEC 48 LET box register, just to the left of the " = " sign.

NOTE: *When adjusting the OUTPUT while under manual control, the user must make the adjustments to the L5 output register in the 18-register block; not directly to the register assigned to the external output. See Example 1 in Section 9.7.*

9.6 LEAD/LAG Functions

NOTE: *This function is also described in Instruction Bulletin #30598-240-0x.*

The SPEC 49 (LEDLAG) instruction in a LET box performs a LEAD/LAG calculation on the process variable (PV). The number preceding SPEC 49 in the LET box is used as a pointer to a user-defined block of seven registers. This number must be an integer from 1 to 3993.

In the following table (Figure 9.2), a "Z" prefix is used to denote the seven storage registers that are used in the LEAD/LAG calculations. Characters in boldface represent variables used in the equations that follow.

REGISTER #	FUNCTION
Z1	The resulting calculation for this lead/lag update ($\mathbf{Q_n}$)
Z2	The process variable used in this update ($\mathbf{PV_n}$)
Z3	The sample interval in milliseconds (T_s)
Z4	The lead time constant in hundredths of a second (T_d)
Z5	The lag time constant in hundredths of a second (T_i)
Z6	The resulting calculation from the previous update ($\mathbf{Q_{n-1}}$)
Z7	The process variable from the previous update ($\mathbf{PV_{n-1}}$)

Figure 9.2 LEAD/LAG Register Allocation

The LEAD/LAG algorithm calculates $\mathbf{Q_n}$ based on user-entered values for T_s, T_d, and T_i. Since these values are stored in integer registers, they contain an implied decimal point that must be entered in the following format:

For T_d and T_i ..... XXX.XX
For T_s ..... XX.XXX

OTHER CONSIDERATIONS

- The process variable (**PV**) must be loaded into register Z2.
- The calculated value of $\mathbf{Q_n}$ must be transferred into the PIDD or PIDR loop register L6 (the process variable register).
- The LEAD/LAG algorithm retains $\mathbf{Q_{n-1}}$ and $\mathbf{PV_{n-1}}$ for the next calculation.
- In order to ensure the integrity of the calculations, Rev. 3 and later revisions of the Model 400 perform limit checks on the variables loaded by the user. Valid ranges are defined as follows:

$0 \le X \le 9999$ for PV.
$X \ge 0$ for T_d and T_i.
$X > 0$ for sample interval T_s.

If any of these range limits are not observed, the LEAD/LAG algorithm is not executed and bit 18 of the destination register (the register to the left of the "=" sign in the LEDLAG rung) is set to flag the error. In addition, the processor sets bit 18 of the out-of-range register to facilitate troubleshooting by identifying the source of the illegal value.

See Example 2 in Section 9.7 for typical ladder rungs required.

The SPEC 49 entry in a LET box causes the following steps to occur:

1. $$Q_n = \left[\frac{\left[\frac{T_i}{100} * \frac{Q_{n-1}}{100} + \left[\left(\frac{T_s}{1000} + \frac{T_d}{100} \right) * \frac{PV_n}{100} \right] \right] - \left(\frac{T_d}{100} * \frac{PV_{n-1}}{100} \right)}{\frac{T_s}{1000} + \frac{T_i}{100}} \right] * 100$$

2. LET $PV_{n-1} = PV_n$ (prepare for next calculation)

3. LET $Q_{n-1} = Q_n$ (prepare for next calculation)

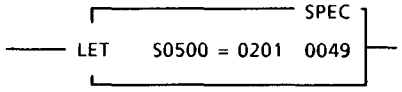
NOTE: *Since the LEAD/LAG algorithm performs a calculation, it is governed by the same rules as any calculation. If the result of the calculation cannot be represented as a 16-bit integer (-32767 to 32767), then numerical overflow errors (refer to Section 7.11 Overflow Errors) will occur. When this happens, the LEAD/LAG calculation is not executed.*

Overflow may occur the very first time the LEAD/LAG function is calculated whenever the lag constant (Z5) set to '0000'. As a strictly lead calculation, the algorithm simplifies as follows:

$$Q_n = PV_n + \left[\frac{10 * T_d}{T_s} * \left(PV_n - PV_{n-1} \right) \right]$$

From this relationship it can be shown that if (T_d/T_s) is greater than or equal to '1', then for PV_n - PV_n-1 values greater than 2978 (approximately 29.78 per cent of scale), numerical overflow will occur. The first time the calculation is performed, PV_n-1 is equal to '0000'. It is also possible for signals with a low signal to noise ratio for overflow to occur periodically during normal ladder scanning.

The LEAD/LAG algorithm in lead only mode is a differentiator. A differentiator is designed to amplify signal, with noise included. The overflow condition can be detected and appropriate application action can be taken.



If overflow occurs in this ladder rung, overflow bit 0500-18 will become energized, and the LEAD/LAG algorithm will not execute. This overflow bit can be used by the application to compensate for the overflow condition.

9.7 Application Considerations and Examples

While the loop algorithms in this section are the *same as shown in* Instruction Bulletins 30598-301-XX and 30598-240-XX, certain differences exist in the way the Model 400 executes and implements the results. For example, although the loop update time is user-set, the Model 400 scan time and I/O update time must be *fast enough to accommodate the loop* time.

The concept of *throughput* must be understood, and the Special Instruction Set for the Model 400 (Section 8 of this manual) must be read and understood as well. A working knowledge of PID operation and a good grasp of the terminology contained in two Instruction Bulletins - *30598-301-0X* and *30598-240-0X* - is recommended before proceeding with this section.

Instruction Bulletin #30598-301-0x, titled Class 8010 PID Closed-Loop Control, Using the SY/MAX Model 300 Programmable Controller, describes the use of a PID control program which can be loaded into a Model 300 or 500 Processor via cartridge tape. The discussions of the PID control equation in Sections 2.0 and 3.1 of that bulletin should be compared to the algorithms for PIDD and PIDR in this bulletin; they are identical except for data on sample interval.

Also, note the similarities between the register usage table in Section 3.2 of Bulletin 30598-301-0X and the usage in the Model 400; loop number, alarm status, and alarm acknowledge registers are not used in the Model 400, but the loop status *does* use several bits. The PV_{n-1}, integral sum Most Significant Digit and integral sum Least Significant Digit registers also differ from the 30598-301-0X bulletin.

Instruction Bulletin #30598-240-0X, titled Class 8040 Type PCM-110 Process Control Module, describes the PID algorithm that is available in the firmware of the PCM. The PID control equation in Figure 8.2 of that bulletin should be compared to the PIDD and PIDR algorithms in this Model 400 bulletin. Note that the PID algorithm A in the PCM bulletin is the same one used for the Model 400.

The register usage table in the PCM bulletin should be compared to the usage of the Model 400. Register differences exist because the PCM, being a dedicated closed-loop controller, has many features not found in the Model 400.

References to alarm status, remote setpoint, deadband, etc., as well as listings beyond loop register 15-do not apply to the algorithm used by the Model 400 (although applying a LEAD/LAG function to the process variable does exist in the form of a SPEC 49 LET box).

The following examples show how some of these special functions are utilized.

> **CAUTION**
>
> **These examples are intended to be used only as guidelines when implementing the algorithms. It is the user's responsibility to understand these examples before incorporating these or any other rungs into a working program.**

EXAMPLE ONE: Reverse-acting single PID loop (PIDR loop)

GIVENS:

- The setpoint (**SP**) and tuning (K_p, K_I, K_D, and **B**) parameters are already set.
- The process variable *input* is in register 513.
- The process variable *output* is in register 517.
- The 18-register PID definition register block consists of registers 51-68.
- Reverse-acting PID control will be applied based on a 200 ms loop update time.
- The loop calculations must be performed at 200 ms intervals. For this example, the necessary repeatability will be achieved with a *timed interrupt* subroutine. Note use of the *timed interrupt precludes* RUN-mode programming. If RUN-mode programming must be preserved, some other technique must be used to ensure execution based on the 200 ms sample interval time.
- If the loop is not in AUTO, loop control will default to MANUAL.

RUNGS NEEDED:

1. TIMED INTERRUPT RUNG - The following rung must be the *very first rung* in the program if a timed interrupt subroutine is used. It allows the main program to call the subroutine containing the PID algorithm at specified intervals. In this example, the subroutine is called every 200 milliseconds with a tolerance of 15 milliseconds. See Section 6.8.1. Note that use of the timed interrupt subroutine does not allow for RUN-mode programming.

2. SAMPLE INTERVAL RUNG - The following rung establishes the sample interval in milliseconds in register 68 (L18). The interval in this example is 200 ms; this value must agree with the value in rung 1, and represents the frequency with which the routine must be executed.

3. HIGH OUTPUT LIMIT RUNG - The following rung programs the high output limit into register 65 (L15). In this example, the maximum high limit of 9999 is programmed.

4. *LOW OUTPUT LIMIT RUNG - The following* rung programs the low output limit into register 66 (L16). In this example, the minimum low limit of 0000 is programmed.

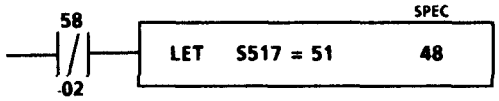
THE MAINLINE USER PROGRAM EXISTS HERE, BETWEEN RUNG "4" ABOVE AND RUNG "A" BELOW

A. SUBROUTINE AREA RUNG - The following rung appears only once in any program, and serves to mark the memory area which holds all subroutines.

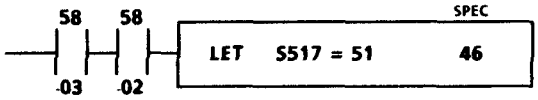
B. START SUBROUTINE RUNG - The following rung marks the point at which the timed interrupt subroutine begins.

C. COPY PV RUNG - The following rung copies the process variable from the external input register (513 in this example) into the loop process variable register (56 in this example, defined as L6 in the register block).

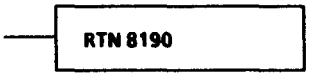
D. EXECUTE PIDM CALCULATION RUNG - The following rung begins PIDM calculations if bit 2 in register 58 has been reset; if bit 2 = 0, then loop is in MANUAL. Register 51 is the pointer that both points to the register block 51-68 and also defines *register 51 as L1, 52 as L2, etc. Register 517 is* this example's external output register, where the calculated output is placed.



E. EXECUTE PIDR CALCULATION RUNG - The following rung begins PIDR calculations if bits 2 and 3 in register 58 have been set. These two bits correspond to the "mode status" bits previously described. Bit 2 set to "1" indicates the loop is NOT in manual, while bit 3 set to "1" indicates the loop is in the AUTO mode. Again, register 51 is the pointer that both points to the register block 51-68 and also defines register 51 as L1, 52 as L2, etc. Register 517 is this example's external output register where the calculated output is placed. To change this loop to a PIDD (direct) loop, only change SPEC 46 to SPEC 47 in the following rung.



F. RETURN FROM SUBROUTINE RUNG - The following rung identifies the end of the timed interrupt subroutine.



EXAMPLE ONE PROGRAMMING CONSIDERATIONS

- The execution time of the timed interrupt subroutine should be kept as short as possible. The rungs that establish the loop initialization constants (for *sample interval, high output limit* and *low output limit*- rungs 2, 3 and 4 in example 1) are programmed in the mainline area of the ladder program.

- Rung C in the subroutine (which transfers the process variable into register L6) is executed immediately prior to the PID rungs (D and E) so that the most recent input update is used in the PID computations.

- Bit 2 of register 58 controls whether the loop is under manual or automatic control. In the Example One program, the loop can be disabled (no PID computations performed) if bit 2 of register 58 = 1 (loop not in MANUAL) and bit 3 of register 58 = 0 (loop not in AUTOMATIC).

- The order in which rungs D and E are programmed is irrelevant.

- The **OUTPUT** for rungs D and E also exists in register 55 (L5). Programming the external output register 517 avoids assigning an additional rung to *LET S517 = S55*.

- If the loop is being executed in MANUAL (bit 2 of register 58 = 0), the output must be adjusted in register 55, *NOT* in register 517. For example, to set the output to 50% of full-scale range, the value 5000 is entered into register 55. When the Model 400 executes rung D, the contents of register 55 are transferred into register 517. If 5000 had been entered into register 517, it would have been overwritten by the current contents of S55 in the transfer operation.

- The worst-case execution time for the block of rungs within the timed interrupt subroutine is approximately one millisecond.

- There is no absolute limit to the number of PID loops that the Model 400 can execute. Practical limitations for a given system are a function of the number of available registers, system throughput, and the user's ability to manage multiple loops. Additional PID loops can be executed by either adding the necessary blocks of rungs into the MARK ST. SUB area or by multiplexing the loop algorithm by changing the pointer from a constant to a variable and the output register from a single address to a matrix. The first method is simple; the second requires experience in advanced programming.

- System throughput is decreased for every PID loop that is added to the program. The interrupt rate must be infrequent enough to allow the Model 400 to scan the remainder of the program in a reasonable amount of time for the application.

- If the PID algorithm fails to execute as expected, check to see if bit 18 of register 517 is set to indicate an out-of-range condition (Rev. 3 or later). If set, monitor the other registers in the block to determine which register is out of range, as indicated by bit 18 of the offending register.

- If the execution time of the timed interrupt subroutine exceeds the interrupt rate, ERROR 29100 is generated and the Model 400 halts program execution.

- If the interrupt tolerance is too restrictive, ERROR 29102 is generated and the processor halts.

EXAMPLE TWO: LEAD/LAG Function Used With Example One Program

NOTE: *Instruction Bulletin 30598-240-XX contains more information on the LEAD/LAG function.*

Assume the same conditions exist as in Example One except that the LEAD/LAG function is to be applied. Additional given conditions for the LEAD/LAG portion of the program are therefore required.

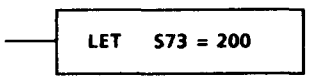
LEAD/LAG GIVENS:

- The process variable exists in register 513.
- Registers 71-77 (Z1-Z7) make up the block that defines the LEAD/LAG function.
- The sample interval T_S is preset in register 73 (Z3).

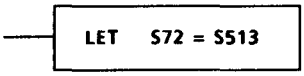
NOTE: *The user must enter the LEAD/LAG time constants (T_D and T_I) in registers 74 (Z4) and 75 (Z5) respectively.*

The following rungs must appear in the timed interrupt subroutine and replace "RUNG C" as shown in the previous Example One in the first example:

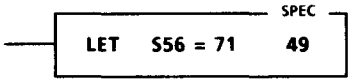
The following rung presets the sample interval (in milliseconds) into register 73 (Z3). This number must correspond to the timed interrupt interval specified in rung 1 and also match the interval specified for the PIDR or PIDD algorithm.



The following rung loads the process variable into register 72 (Z2) for use in the LEAD/LAG calculation.



The following rung performs the LEAD/LAG function on the process variable currently loaded in register 72 (Z2). "71" is the pointer that points at registers 71-77. The calculated Q_n is placed in register 56 for use in the upcoming PIDM or PIDR calculations.



10 ASCII COMMUNICATIONS (Input and Output)

10.1 Description/Initiation

The Model 400 uses an enhanced PRINT rung format to allow inputting of ASCII-coded data. Termination of the ASCII input string can be done either by character *count*, character *delimiter*, or both.

The ASCII input programming format is similar to that of ASCII output. A special register address (S8006) is inserted as the first register in the PRINT statement and is used to identify the mode. Subsequent entries in the PRINT statement further define the function.

The ASCII input rung must be programmed *with all entries* as shown in Figure 10.1:

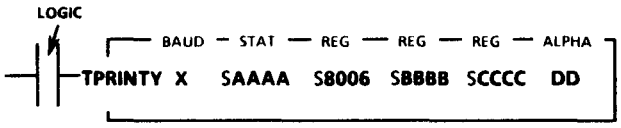


Figure 10.1 ASCII Input Rung

Each rung parameter in Figure 10.1 is explained as follows:

LOGIC upon becoming TRUE, the port specified by Y is enabled to accept ASCII input characters. When FALSE, the port ignores the ASCII input and reverts to SY/MAX protocol communication, unless a complete ASCII input string had not been received when previously TRUE. Refer to Section 10.3.

Y is the CHANNEL that receives the input, being either 1 (PRGMR port) or 2 (COMM port). The serial word structure for channel 1 is defined in register 8099. The structure for channel 2 is in register 8100. See Figure 10.3.

X is the BAUD RATE for the selected channel (Y). See Figure 10.4 for available baud rates.

SAAAA is the STATUS REGISTER assigned to this communication rung. DO NOT use this user-specified register for any other purpose in the program.

Error codes that may appear in the status register are listed in Section 10.5.

S8006 is not actually a register, but is the COMMAND that defines the box as an ASCII input box.

SBBBB identifies the STARTING REGISTER for storing the data block. The block can hold no more than 256 ASCII characters. If more than 256 characters are input, an error is sent to the status register and ERROR 19 is posted. *This register is the start address, not a pointer to a start address.* Each register in the block will contain a single character (stored in the least significant byte).

SCCCC identifies the register whose contents specify the number of characters to be inputted. When this number of ASCII characters has been received, the ASCII input string is terminated and bit 16 of the associated status register is set ON.

DD are the contents of the ALPHA box that identifies the ASCII character which the user specifies as the *block terminator* (delimiter). The desired terminator character should be loaded, followed by a space (space bar) character to shift the character into position. When this character has been received, the ASCII input string is terminated and bit 16 of the associated status register is set ON.

CAUTION

At the end of every block of received data, the Model 400 appends a null (00) character to mark the end of the block. Due to this feature, the user MUST reserve one additional register beyond the maximum character count for the block size. If insufficient registers are reserved, incoming data may overwrite register data that is used for other purposes in the program.

NOTES:

1. *If no characters are entered for* **DD** *in the ALPHA box, the default block terminator is a space (20H).*
2. *Setting the contents of* **SCCCC** *to zero defaults to the maximum input block size of 256 characters.*
3. *If both* **SCCCC** *and* **DD** *are specified, the ASCII input block will stop upon reaching the first condition to be satisfied.*

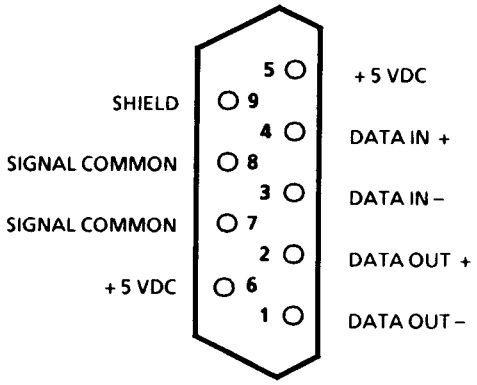


Figure 10.2 PRGMR and COMM Port Pinout

10.2 Port Configuration

Both the PRGMR and COMM ports of the Model 400 are configured as RS-422 ports. The pinout for these ports is shown in Figure 10.2.

NOTE: *RTS and CTS hardware handshaking signals are not supported. Pins 5, 6, 7, and 8 are used for + 5VDC and signal commons.*

When configuring each of the PRGMR and COMM ports for ASCII input, registers 8099 (for PRGMR) and 8100 (for COMM) are used. Figure 10.3 illustrates what each of the bits in these two registers are used for and how to configure the port for the desired word structure.

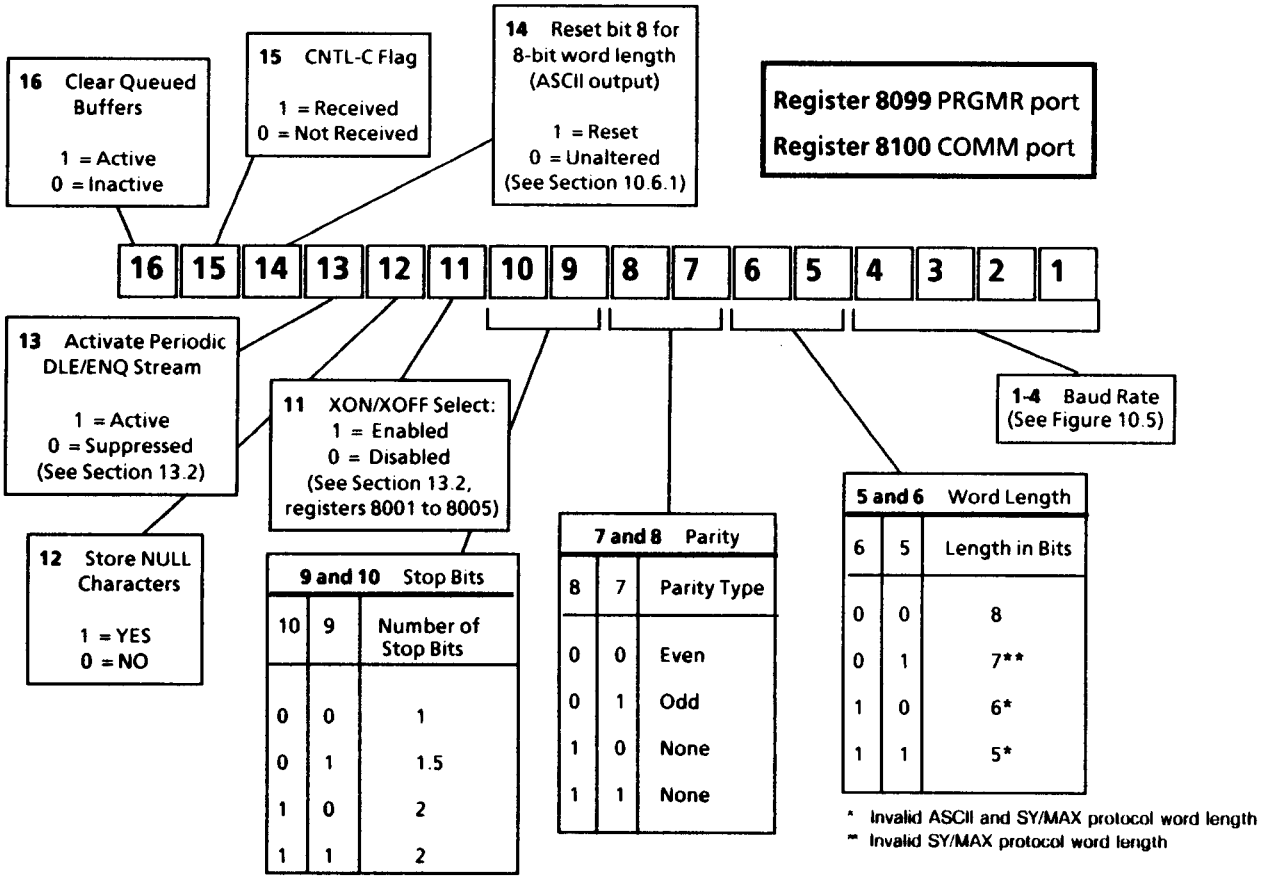


Figure 10.3 PGRMR and COMM Set-up Registers

10.3 Operation

When the logic preceding the ASCII input box transitions from FALSE to TRUE, the specified port is enabled to receive ASCII. Once enabled, the port remains enabled to accept an ASCII input string even if the logic preceding the ASCII input box becomes FALSE.

Incoming characters are stored in a communication port buffer until a complete ASCII string has been received. As soon as the final character is received, the port buffer ignores future ASCII characters. These characters are lost unless another port buffer has been queued to accept them. A total of 11 buffers exist for both ports. Of these, up to 9 can be allocated by the processor to a single port. This makes it possible to anticipate multiple ASCII input strings by enabling multiple ASCII input rungs, or even by repeatedly transitioning the logic of a single rung, since a new buffer is allocated upon every transition.

The port buffer is then read by the processor during its end-of-scan communication port servicing and the ASCII characters are moved into the registers specified in the ASCII input rung.

Bits in the status register reflect the above situations; bit 22 is set to "1" when the input logic is TRUE, bit 17 is set when a port buffer is enabled for input, and bit 16 is set when a complete ASCII input string has been received and processed by the CPU. Figure 10.4 shows which status bits are set based on the ASCII input rung's current logical state and whether a port buffer is queued for ASCII input from a *previous* FALSE-to-TRUE logic transition.

LOGIC	BUFFER QUEUED	PORT ASCII INPUT STATUS	RESULTING BIT STATUS		
			BIT 16	*BIT 17*	*BIT 22*
FALSE	NO	INPUT DISABLED	0	0	0
TRUE	YES	INPUT ENABLED	0	1	1
TRUE	NO	INPUT COMPLETE	1	1	1
FALSE	YES	INPUT ENABLED	0	0*	0

** When LOGIC is FALSE, status bits are cleared but buffer remains in queue.*

Figure 10.4 Bit Status vs Logic and Buffer States

BAUD RATE CODE		
BINARY BITS 4321	DECIMAL	BAUD RATE
0000	0	75
0001	1	110
0010	2	300
0011	3	1200
0100	4	2400
0101	5	4800
0110	6	9600
0111	7	19 2K

Figure 10.5 Baud Rate vs Bit Pattern (See Figure 10.3)

Buffers become "linked" with the ASCII input rungs in the order in which they are executed TRUE, *unless all 9 buffers have been allocated.* In this case, even though status register bit 22 is set, bit 17 *is not.* This indicates that all buffers are queued. A buffer is not freed up until a complete ASCII string has been received and the buffer has been read by the Model 400 during its end-of-scan activities.

NOTE: *If all 9 buffers are simultaneously used for ASCII input while priority SY/MAX communications (READ, WRITE, ALARM) are transmitting out the other port, port 2 (COMM) must be used for ASCII input; port 1 (PRGMR) is used for the SY/MAX communication.*

10.4 Application Considerations

- ASCII storage registers contain the binary code for the ASCII character, not the character itself. For example, the ASCII character "0" is represented as "48" (decimal), where an ASCII "A" is represented as "65" (decimal).

- One ASCII character is stored per register (in the least significant byte).

- ASCII input information does not appear in the storage registers until a complete block, defined by the block size or the terminator, has been received.

- Null (00) characters may either be disregarded or stored as valid characters (Rev. 2 or later only). See bit 12 in Figure 10.3 .

- If null characters are disregarded, the end of the input register block is defined either by the block terminator or by the first register to contain a null character.

- Revision 2 and later versions of the Model 400 have added functions to bits 15 and 16 of port attribute registers 8099 and 8100.

  Bit 16 - when set, aborts any pending ASCII input instructions and clears any port buffers queued for an ASCII input string. This is a means of error recovery when characters in an ASCII input string are lost or garbled. Setting bit 16 to "1" gives the user a means of purging the input buffers in preparation for a new ASCII string.

  Bit 15 - acts as a flag that recognizes the ASCII input BREAK command. Upon receiving a CTRL-C (03H), bit 15 latches ON. For example, when programmed as a contact to control bit 16, a CTRL-C character issued from the ASCII output device purges and resets any queued ASCII input buffers.

- Both bits 15 and 16 of the port attribute registers are reset to "0" upon power-up.

- Upon a power cycle, port 1 (PRGMR) resets to 9600 baud, 8-bit word, even parity, with one stop bit in order to be compatible with the SY/MAX protocol. Port 2 (COMM) retains any changes made to its attributes.

- DEL/CLR ALL does *not* cause port attributes to resets to the default values *unless* prompted by a "CLR MEMORY NEEDED".

- Because valid ASCII characters are defined as seven bits long, the Model 400 (when configured to receive an *eight*-bit ASCII word) masks off the state of bit 8 in the incoming character.

- Halting the Model 400 or setting bit 16 of register 8099 or 8100 to "1", while awaiting an ASCII input string, clears any queued buffers, halts the ASCII input instruction, and generates ERROR 17 in the status register.

- The ASCII input function only operates properly when the device that is outputting the ASCII characters is *directly* connected to a Model 400 port. ASCII sent over the network is not accepted by the ASCII input function.

- When a port is queued for ASCII input while FORCING is active, the "ASCII port" is assumed to be dedicated to the ASCII input function. If the cable is removed from the *other* communication port, no DLE/ENQ characters are transmitted out the ASCII port to inquire if a programming device is present.

- If a port's word structure is altered, it *remains* altered even after the port reverts to SY/MAX protocol. SY/MAX protocol requires an 8-bit word with even parity, and one stop bit, to operate correctly..

10.5 Error Codes

The error codes shown in Figure 10.6 below apply to the ASCII input operation and may appear in the status register of the ASCII input rung ("SAAAA", in Figure 10.1).

ERROR NUMBER	DESCRIPTION
11	Receiver overflow. The Model 400 is unable to handle incoming characters fast enough, due to the servicing of other interrupts.
13	Parity error
15	Framing error. Typically due to baud rate or word structure problems.
17	Input was terminated before a complete ASCII string was received. Results from either the CPU going to HALT or bit 16 of register 8099 or 8100 being set. If bit 16 is set, buffers queued for ASCII input are cleared.
19	Buffer overflow. The 256-character buffer was filled before the terminating character was received.

Figure 10.6 ASCII Input Error Codes

10.6 ASCII Output

The capability for outputting ASCII characters via PRINT statements exists for both Model 400 ports. Refer to the appropriate programmer instruction bulletin 30598-167, 30598-174 or 30598-717 for general information on programming PRINT rungs.

10.6.1 STANDARD ASCII FORMAT

The default ASCII characters are output as RS-422 signals (see Figure 10.2 for port pin-out) defaulting to 8-bit words with even parity and 1 stop bit, transmitted at 9600 baud.

NOTE: If the receiving device does not automatically mask off bit 8 (typically indicated when half the characters are received correctly, the other half garbled), set bit 14 of register 8100 for port 2 (register 8099 for port 1) to 1 to cause the processor to perform the mask. Another alternative to match output default settings is to set up the receiving device for a 7-bit word with odd parity and 2 stop bits. Other settings may be accommodated by adjusting registers 8099 or 8100; refer to Figure 10.3.

Standard PRINT statements allow outputting fixed ASCII characters (represented by ALPHA entries in the PRINT rung) and register contents (represented by REG entries in the PRINT rung). This allows an English language message to be augmented with variable register data. Each REG entry will result in 6 ASCII characters being transmitted, representing up to 5 decimal places plus sign (if negative) or space (if positive). Leading zeros will automatically be transmitted as spaces. Each standard ASCII PRINT statement automatically appends a CR and LF to the ASCII string, unless suppressed with a semi-colon(;) in the last ALPHA position.

10.6.2 RAW BINARY ASCII FORMAT

While the standard ASCII format will handle the majority of ASCII output applications, some of the special CONTROL CODES (ESC, DLE, etc.) cannot be accommodated by the ALPHA entries. Any 7-bit ASCII code (0 through 127) can be output via the Raw Binary ASCII format.

To invoke the Raw Binary format, utilize the same PRINT statement as for the standard format with 2 exceptions: REG S8004 must be the first element following STAT in the horizontal PRINT box (refer to Figure 10.7 below) and all subsequent elements of the box should be REG, not ALPHA.

To output the ASCII codes, store the desired 8-bit patterns in the registers specified by the REG elements. The data in each register will be treated as a pair of 8-bit characters (low-byte bits 1-8, high-byte bits 9-16) with each byte ranging from 0 through 127. The high-byte character is transmitted *first, followed by the low-byte character. If the* contents of the register is a single byte (0-127), then that byte, representing a single ASCII character, is transmitted.

EXAMPLE: This example demonstrates the use of register 8004 in a print rung to set the "raw binary" output format.

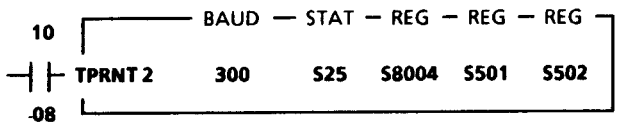


Figure 10.7 Raw Binary Example

To output a carriage return (CR), a line feed (LF), and the number "1", three bytes are required. This requires assigning two registers - the first register contains the CR in the high byte and the LF in is the low byte. The second register's low byte contains the number "1". Registers 501 and 502 are used in this example to contain the ASCII codes.

A suggestion on how to preset these values is to multiply the desired high byte of register 501 (the CR is ASCII 13) by 256 then add the low byte (the LF, ASCII 10) to the product. This yields 3338 which should be preset in register 501. This leaves the single byte (the number "1"); preset its ASCII value (49) directly in the low byte of register 502.

When the contact 10-08 is closed in the above rung, a CR followed by a LF followed by the number "1" will be sent out of port 2 at 300 baud.

10.6.3 MISCELLANEOUS ASCII FORMATS

Up to this point, the REG elements in a PRINT box could be output as a standard decimal value (5 digits plus sign in the standard ASCII format) or as an ASCII character representation (in the "raw binary" ASCII format). It is also possible to print out the register contents in 2 other formats: a 16-digit binary representation of the register bit pattern (all characters either "0" or "1") or a 4-digit hexadecimal representation (characters 0 through F).

To invoke the 16-digit binary representation, have REG S8002 be the first element following STAT in the horizontal PRINT box; for the 4-digit hexadecimal representation, use REG S8003 as the first element. All subsequent REG elements of the PRINT box will be output in the respective format.

NOTE: *To cause a PRINT box format to revert to the default of 5 digits plus sign, replace S8002 or S8003 with S8001.*

10.6.4 REPEATED CHARACTER STRINGS

For applications which require outputting the same character numerous times, another special format is available. By having REG S8005 as the first element following STAT in the horizontal PRINT box, subsequent REG elements will be interpreted such that the low byte represents the ASCII character (0-127) and the high byte the number of times this character is to be repeated (up to a maximum of 192). If the high byte = 0 or is greater than 192, no data is sent.

10.6.5 XON/XOFF HANDSHAKING

For all PRINT formats, the Model 400 can be enabled to support XON/XOFF software handshaking while outputting ASCII. Setting bit 11 of register 8099 (PRGMR port) or 8100 (COMM port) to 1 allows a receiving device (a printer, for example) attached to the respective port to instruct the Model 400 to stop sending ASCII. The device does this by sending a "DC3" (13H or CTRL-S) to the Model 400. When the receiving device is again ready to accept ASCII, it sends a "DC1" (11H or CTRL-Q) to the Model 400.

NOTE: *When XON/XOFF is active while SY/MAX communications (READ, WRITE, ALARM) are being transmitted out the other port, port 2 (COMM) must be used for the ASCII function; port 1 (PRGMR) must therefore be used for the SY/MAX protocol device.*

10.7 ASCII Hex/Decimal Conversion

Figure 10.8 provides a conversion table for ASCII characters in hexadecimal and decimal numbers.

ASCII Character	VALUE Decimal (Hex)	ASCII Character	VALUE Decimal (Hex)	ASCII Character	VALUE Decimal (Hex)	ASCII Character	VALUE Decimal (Hex)
NUL	00 (0000H)	Space	32 (0020H)	@	64 (0040H)	'	96 (0060H)
SOH	01 (0001H)	!	33 (0021H)	A	65 (0041H)	a	97 (0061H)
STX	02 (0002H)	"	34 (0022H)	B	66 (0042H)	b	98 (0062H)
ETX	03 (0003H)	#	35 (0023H)	C	67 (0043H)	c	99 (0063H)
EOT	04 (0004H)	$	36 (0024H)	D	68 (0044H)	d	100 (0064H)
ENQ	05 (0005H)	%	37 (0025H)	E	69 (0045H)	e	101 (0065H)
ACK	06 (0006H)	&	38 (0026H)	F	70 (0046H)	f	102 (0066H)
BEL	07 (0007H)	'	39 (0027H)	G	71 (0047H)	g	103 (0067H)
BS	08 (0008H)	(	40 (0028H)	H	72 (0048H)	h	104 (0068H)
HT	09 (0009H)	)	41 (0029H)	I	73 (0049H)	i	105 (0069H)
LF	10 (000AH)	*	42 (002AH)	J	74 (004AH)	j	106 (006AH)
VT	11 (000BH)	+	43 (002BH)	K	75 (004BH)	k	107 (006BH)
FF	12 (000CH)	,	44 (002CH)	L	76 (004CH)	l	108 (006CH)
CR	13 (000DH)	-	45 (002DH)	M	77 (004DH)	m	109 (006DH)
SO	14 (000EH)	.	46 (002EH)	N	78 (004EH)	n	110 (006EH)
SI	15 (000FH)	/	47 (002FH)	O	79 (004FH)	o	111 (006FH)
DLE	16 (0010H)	0	48 (0030H)	P	80 (0050H)	p	112 (0070H)
DC1	17 (0011H)	1	49 (0031H)	Q	81 (0051H)	q	113 (0071H)
DC2	18 (0012H)	2	50 (0032H)	R	82 (0052H)	r	114 (0072H)
DC3	19 (0013H)	3	51 (0033H)	S	83 (0053H)	s	115 (0073H)
DC4	20 (0014H)	4	52 (0034H)	T	84 (0054H)	t	116 (0074H)
NAK	21 (0015H)	5	53 (0035H)	U	85 (0055H)	u	117 (0075H)
SYN	22 (0016H)	6	54 (0036H)	V	86 (0056H)	v	118 (0076H)
ETB	23 (0017H)	7	55 (0037H)	W	87 (0057H)	w	119 (0077H)
CAN	24 (0018H)	8	56 (0038H)	X	88 (0058H)	x	120 (0078H)
EM	25 (0019H)	9	57 (0039H)	Y	89 (0059H)	y	121 (0079H)
SUB	26 (001AH)	:	58 (003AH)	Z	90 (005AH)	z	122 (007AH)
ESC	27 (001BH)	;	59 (003BH)	[	91 (005BH)	{	123 (007BH)
FS	28 (001CH)	<	60 (003CH)	\	92 (005CH)	|	124 (007CH)
GS	29 (001DH)	=	61 (003DH)	]	93 (005DH)	}	125 (007DH)
RS	30 (001EH)	>	62 (003EH)	^	94 (005EH)	~	126 (007EH)
US	31 (001FH)	?	63 (003FH)	_	95 (005FH)	DEL	127 (007FH)

Figure 10.8 ASCII Hex/Decimal Conversion

11 SEQUENCER

11.1 Description and Initiation

The Model 400 processor is equipped with a Sequencer function that allows the output of information based on time driven and/or event (input) driven conditions.

OPERATIONAL ATTRIBUTES

- Up to 255 steps can be defined in a sequence.
- Up to 64 outputs can be controlled.
- Stepping can be performed forward, reverse, or non-sequentially.
- Automatic or manual operation is allowed.
- An automatic step advance can be based on time, events (up to 16 inputs), or both.
- Time accumulated during a step can be held (paused).
- Output state change can be inhibited despite step completion.
- Individual step times can be programmed with a resolution of 0.1 second.
- A sequence can be repeated, or halted after a single pass.
- As many Sequencers can be programmed as allowed by register availability.

The Sequencer is initiated via the SPEC 65 function command. This command points to a block of registers which contain the operating parameters of the Sequencer.

The minimum rungs needed to program *one* Sequencer are shown in Figure 11.1.

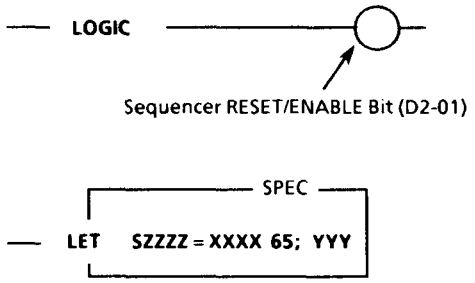


Figure 11.1 Sequencer Initiation Rungs

The rung variables in Figure 11.1 are:

LOGIC Logic in the first rung allows the Sequencer to be reset, or enabled to operate.

SZZZZ A dummy output register. The data in this register will always be the current step number. The recommended programming parameter is: *ZZZZ = XXXX*.

XXXX A value that points to the definition register block. This value must be a constant and range from 1 to 3985.

SPEC 65 SPECial Instruction #65. Specifies the Sequencer function.

YYY The number of steps in the Sequencer. Must be a constant from 1 to 255.

NOTE: *Placing logic in front of the LET box could result in the Sequencer not being executed.*

11.2 Register Block Allocation

The overall length of each Sequencer register block is variable and changes depending on the number of steps programmed into the Sequencer.

The first ten registers in the block are always the *configuration table registers* and serve to define conditions for *all* steps. Each step also requires six registers to define itself.

Figure 11.2 illustrates how the Sequencer registers are allocated. Since each step used within the Sequencer requires six registers, the total number of registers required per Sequencer are calculated as follows:

[6 × (number of steps) + 10]

See Section 11.2.1 for information on the configuration registers and 11.2.2 for the step registers.

NOTE: *The "D" prefix is intended to convey the idea that the register numbers in Figure 11.2 represent relative positions – NOT fixed register numbers.*

11.2.1 CONFIGURATION TABLE REGISTERS

The first ten registers in each Sequencer register block are the configuration table registers. Their function, which affects all steps in the Sequencer, is described in the table shown in Figure 11.3.

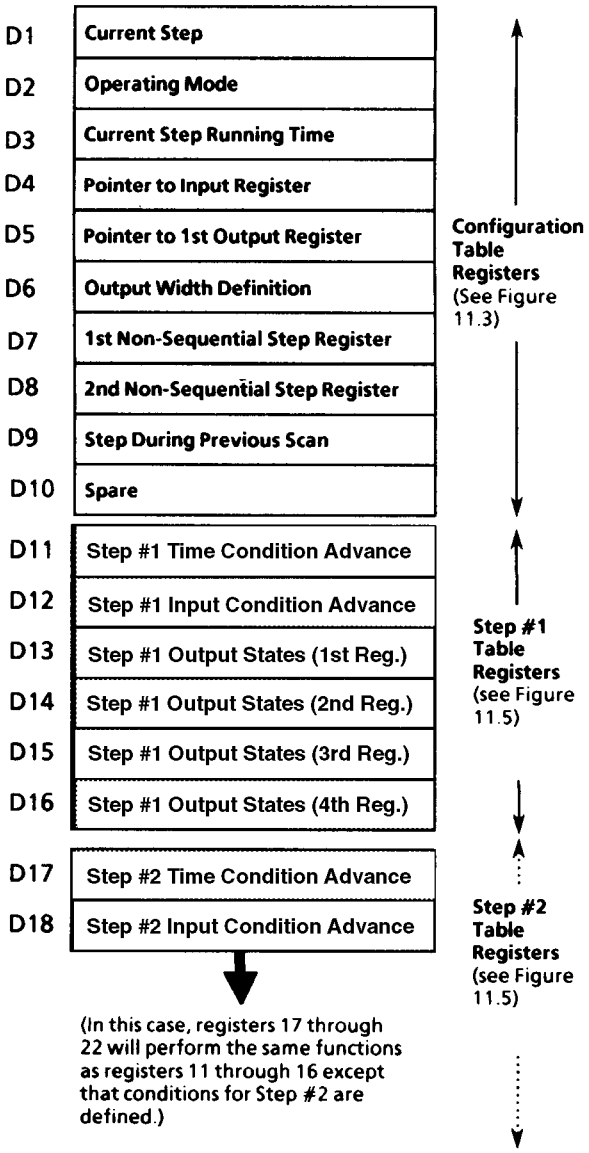


Figure 11.2 Register Allocation

REGISTER #	PURPOSE
D1	*Current step number*. Indicates which step is presently executing. If changed by the user, the sequencer will *immediately* go to the indicated step.
D2	*Operating mode* register. This register defines the sequencer's operating conditions. See Figure 11.4 for a definition of each bit in this register, and Sections 11.4.1 and 11.4.2 for descriptions of the status bits.
D3	*Amount of time* that the current step has been executing. When this register is greater than or equal to the "step time" condition advance (D11, D17, etc), a step advance based on time is enabled.
D4	*Pointer to the input register*. Bits in the register being *pointed to* will be compared with the "step input" condition advance (D12, D18, etc.). When the pattern of specified "1"s is matched, a step advance based on input conditions is enabled. If contents are zero, sequencer operates strictly on time condition advance.
D5	*Pointer to the first output register*. The *contents of this register point to (define)* the first register address of the output field. Bits from the first register of the respective "step output" states (D13, D19, etc.) will be written to the register being pointed at by the contents of D5.
D6	*Bit width of output data*. Valid range is from 1 to 64. This register defines how many bits (beginning with the output register defined by register D5) make up the output field. A value greater than 16 will begin with the low-order bits of the next consecutive register; for example, if register 21 is pointed to as the first output register, and output width = 18, sequencer outputs will be written to register 21 *and* the first 2 (least significant) bits of register 22. Bits 3-16 of register 22, along with registers 23 and 24, are unaffected.
D7	*Non-sequential step register*. Setting bit 8 of the operating mode register D2 causes the sequencer to jump to the step loaded by the user in this register. Note that this jump occurs when conditions for advancing out of the current step are satisfied.
D8	*Same as D7* except that D8 works in conjunction with *bit 9* of the operating mode register D2. Registers D7 and D8 function identically; two registers are provided for flexibility and ease of use, specifically to avoid software timing problems or race conditions.
D9	Used by the Model 400 to keep track of the step executed during the previous scan. If this register is not equal to the current step *register (D1), the sequencer immediately* goes to the new step pointed at by D1.
D10	Spare (not presently used)

Figure 11.3 Configuration Table Registers

REGISTER D2 BIT #	PURPOSE
1	**Reset/Enable Sequencer** ("1" = ENABLE, "0" = RESET) When initially enabled, sequencer begins with step #1. Outputs pointed to by registers D5 and D6 will assume the state dictated by Step Register D13 (and D14–D16 if D6 ≥ 17).
2	**Sequence Step Direction** ("1" = reverse, "0" = forward)
3	**RUN/PAUSE select** ("1" = pause, "0" = run) When this bit = 0, elapsed time for a step is allowed to accumulate. When bit = "1", time will *not* accumulate.
4	**Automatic Step Advance Inhibit**. ("1" = hold data, "0" = advance step) When bit = 0, outputs will assume the next state when conditions for an *automatic step advance have been* satisfied. When bit = 1, outputs will retain their previous state whether or not the automatic step advance conditions have been satisfied.
5	**AUTO/MANUAL Select** ("1" = manual operation, "0" = automatic operation) When in AUTO mode, sequencer is controlled by time/input advance *information. When in MANUAL,* sequencer advance is controlled by bit 7.
6	**REPEAT/SINGLE PASS Select** ("0" = single-pass, "1" = auto recycle) When bit = 0, sequencer will execute the sequence one time. At the conclusion of the sequence (the last step), all outputs will reset and the "current step" register D1 will indicate "0". When bit = 1, sequencer will proceed to step #1 after meeting all required conditions for advancing beyond the last step.
7	**Manual Advance** ("1" = advance, "0" = maintain) If bit #5 = 1, toggling this bit from "0" to "1" will advance the sequencer.
8	**Sequential/Non-sequential** ("1" = next step not sequential, "0" = next step sequential) When set to "1", the next step advance will be to the step indicated in D7.
9	Same as bit 8, except that the next step is indicated in register D8.
10	**Time and/or Input condition Advance** ("1" = OR, "0" = AND). When "0", step advance is enabled when input condition is satisfied ONLY AFTER the time advance condition has been met. When this bit = "1", the step advance is enabled when *either* the time *or* input condition has been satisfied.
11–16	Spares (not presently used)

Figure 11.4 Operating Mode Register (D2) Bit Definition (column 1 of 2)

REGISTER D2 BIT #	PURPOSE
17*	Sequence completed.
18*	ERROR bit. See Section 11.4.1.
19*	Manual advance history
20*	Time advance enabled.
21*	Input advance enabled.
22*	Reset/enable history
23*	Step advance just occurred.

* Read-Only Status Bit - See Sections 11.4.1 and 11.4.2

Figure 11.4 Operating Mode Register (D2) Bit Definition (column 2 of 2)

11.2.2 STEP TABLE REGISTERS

The *step table registers* apply only to a particular step in a sequence. The fixed size for each step table register block is six (see Figure 11.2).

Since the step registers begin immediately after the configuration registers, the first block of step registers occupy the 11th through the 16th register in the Sequencer's overall block of registers. Additional step table register blocks occupy the 17th through the 22nd, the 23rd through the 28th, etc as needed.

Figure 11.5 shows the purpose for each of the six step table registers.Each block of step table registers contains the same parameters – i.e., register 11 defines the same parameter for the first step register block as does register 17 for the 2nd step register block. Similarly, register 12 is the same as register 18, register 13 matches register 19, etc.

REGISTER #	PURPOSE
D11/D17/...	**Time Condition Advance Register**. Contains the user-specified step time of this step. Entered in 0.1 second increments. When this time is less than or equal to the Current Step Running Time Register D3, a step advance based on time is enabled. If this register = 0, then the step is strictly input-driven.
D12/D18/...	**Input Condition Advance Register.** Indicates the bits that need to be set before a step advance based on an input condition is enabled. If this register = 0, then step advance is based solely on time. If a step advance is contingent on both time and inputs, normal sequencer operation is such that a comparison for inputs will not be made until *after* the time condition has been satisfied (modifiable by bit 10, register D2).
D13/D19/...	**Defines the output states** of the first output register for each respective step. These bits are written to the "pointed at" register from the *first* output register for the step currently being executed. **NOTE:** *Either register D11/17/... or D12/18/... must contain non-zero numbers before this register is relevant. If both of the conditional advance registers = 0, the step is advanced through.*
D14/D20/...	Same as register D13/D19/..., except that these are the output states for the *second* register (bits 17-32). This register is only relevant if output width (D6) ≥ 17.
D15/D21/...	Same as register D13/D19/... except that these are the output states for the *third* register (bits 33 to 48). This register is only relevant if output width (D6) ≥ 33.
D16/D22/...	Same as register D13/D19/... except that these are the output states for the *fourth* register (bits 49 to 64). This register is only valid if output width (D6) ≥ 49.

Figure 11.5 Step Table Registers

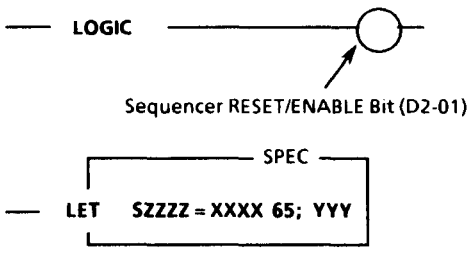
11.3 Application Discussion

The Sequencer provides flexibility for tailoring Sequencer operation to specific needs. This translates into developing a basic sequential algorithm which can then be modified to perform either semi- or non-sequentially. Up to this point the functions of the registers and control bits have been mechanically presented. The purpose of this section is to discuss, from an application standpoint, how to put all of these bits and registers to work in practical situations.

11.3.1 REQUIRED ENTRIES

Regardless of how the Sequencer will be used, the following are *required* actions.

1. The two rungs shown in Figure 11.1 *must* be entered. Since the LET rung must be executed every scan (even when the Sequencer is reset), there must not be any logic in front of the LET rung itself.



Sequencer Initiation Rungs

2. When programming the LET rung, the *user-defined* block of registers that regulates the Sequencer must be entered correctly. Whatever register is specified to the left of the "=" sign contains the current step number. It is suggested that "ZZZZ" be equal to "XXXX" so that the Sequencer can be located in the program by SEARCHing for the first register in the block.

3. The rung containing the RESET/ENABLE bit should precede the LET rung. Other rungs can then be used to preset register values or control bits.

4. The following registers are part of the basic Sequencer set-up procedure and *must* be included in the Sequencer program:

   - **REGISTER D2, BITS 2 through 10.** Note the default here is all "0"s, meaning forward step/single-pass automatic operation.

   - **REGISTER D4.** Points to the register that is *used to determine input conditions for* advance. If sequence advance is based solely on time, register D4 should equal "0".

   - **REGISTER D5.** Points to the first output register written to by the step output states.

   - **REGISTER D6.** Identifies the total number of output bits occupying contiguous registers beginning with the register pointed at by register D5.

   - **REGISTERS D11** (1st step), **D17** (2nd step), **D23** (3rd step), etc. For each step, specifies the amount of time to be spent in the step prior to enabling a time-based advance. If the step advance is based *solely on input conditions*, these registers should all be zero.

   - **REGISTERS D12** (1st step), **D18** (2nd step), **D24** (3rd step), etc. For each step, specifies the binary pattern of "1"s that, when matched by the binary pattern in the input register pointed at by D4, enables a step advance based on inputs. If the step advance is based *solely on time*, the specified binary pattern in these registers should be all "0"s.

   - **REGISTERS 13 through 16** (1st step), **19** through **22** (2nd step), **25** through **28** (3rd step), etc. Output bit patterns that are user-created for each step, in accordance with the output field width indicated by register D6.

11.3.2 OPERATING CHARACTERISTICS

The Sequencer executes during each scan of the ladder program to assure program control of the outputs based on user-defined status. The Sequencer operates as follows:

1. When the Sequencer is RESET (bit 1 of the Operating Mode register = 0), all outputs in the program that are specified by registers D5 and D6 in the Sequencer are turned OFF. Current Step register (D1) is reset to zero. Current Step Running Time register D3 is also reset to zero.

2. *When the Sequencer is* ENABLED (bit 1 *of* Operating Mode register ="1"), the specified output states of step 1 (defined by registers D13 through D16) are transferred to the outputs defined by both the pointer register D5 and output width register D6. The Current Step Running Time register begins to accumulate time.

3. Step conditions for the next advance contained in registers D11 and D12 are compared with the real-time states of the Current Step Running Time register D3 and with the bit pattern of the input register being pointed at by register D4. Bit 10 of the Operating Mode register is checked to determine if either or both conditions must be met prior to a step advance. The Current Step Running Time register accumulates time as long as bit 3 of the operating mode register is not set to "1".

4. During every scan, in conjunction with the above comparisons, the Sequencer checks the status of bit 5 in the Operating Mode register D1. This is a check for AUTO or MANUAL operation. If MANUAL is set, the *time* and *input* advance parameters become meaningless since step advance is based solely on bit 7 being toggled from "0" to "1".

5. Whether in AUTO or MANUAL, the Sequencer now knows what criteria to monitor when determining a step advance.

6. If the Sequencer is in AUTO and the appropriate criteria for a step advance are satisfied, bit 4 of register D2 is checked to see if advance is inhibited.

7. If bit 4 = "0" (advance *not* inhibited), bits 8 and 9 of register D2 are checked to see if the advance is non-sequential.

8. If the advance *is* sequential, bit 2 of register D2 is checked to determine if the advance is forward (incrementing) or reverse (decrementing).

9. The Sequencer checks to see if it has just completed a pass through the final step. If it has, then bit 6 of register D2 is also checked.

10. Finally, if a step change is warranted, the output states for the appropriate step are written to the outputs being pointed at. The entire process then re-starts.

The values of these Sequencer control registers and bits can be changed by the user at any time.

NOTE: *If the Current Step register (D1) is suddenly changed to a different valid step number, the new step is implemented the next time the Sequencer rung is scanned. New output states are written, Current Step Running Time is reset to "0". New Time Advance and/or Input Advance conditions associated with the new step apply.*

11.3.3 TYPICAL OPERATING MODES

This section presents insights into setting up the Sequencer for execution of some typical configurations. The focus here is on manipulating bits within the Operating Mode Register (Register D2, described in Figure 11.4).

As discussed in Section 11.3.1, registers 4, 5 and 6 of the *Configuration Table* indicate the Input Pointer, Output Pointer, and Output Width. These three registers, along with all the *Step Table* registers for Time Advance, Input Advance, and output states of each step, provide the necessary information for Sequencer operation. Normally, these registers once set, remain set in that pattern. However to guarantee the set pattern, it is advised to program additional LET rungs while configuring the Sequencer.

Although only two rungs are mandatory for Sequencer initiation (Figure 11.1), it is a good idea to use as many additional LET rungs in the program as practical. Using "LET presets" to set Sequencer values eliminates the chance of the values being lost, since the values are preserved in the ladder program.

A good practice is to locate the LET presets in the beginning of the program in conjunction with a GOTO that bypasses these statements after the Sequencer is initialized. This bypassing procedure eliminates the time penalty that redundant and unnecessary scanning of the LET rungs incurs.

Of course, the *Operating Mode register* bits can also be programmed into the ladder program. However, rungs containing these bits *must be executed every scan.* To insure proper set-up, the Operating Mode register rungs should be located between the two mandatory rungs as shown in Figure 11.6.

The Sequencer is now assumed to be properly set up. Some common Sequencer operating modes are explained in the following and include TIME, EVENT, TIME/EVENT COMBINATION, and MANUAL sequencing.

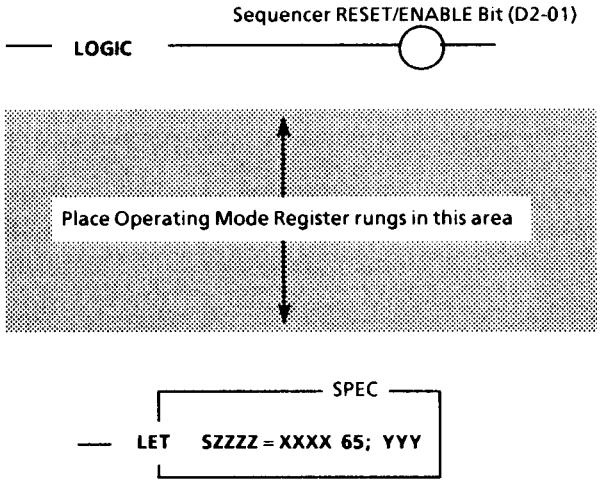


Figure 11.6 Operating Mode Rungs

TIME SEQUENCING ONLY

The following parameters apply for a time-advanced Sequencer:

- Operation is *automatic;* bit 5 of register D2 = "0".
- Since no Input Advance condition is used, set register D4 (and D12/D18/etc., depending on the number of steps) to "0".
- Enter the Time Advance condition into register D11 (and D17/D23/etc., depending on the number of steps). The time is entered in tenth-second (0.1 second) increments.
- Maximum time value for a single step is 32,767 (3,276 seconds, or 54 minutes and 36 seconds).

OPERATION: When bit 1 of register D2 is set to "1", the output pattern associated with Sequencer step #1 (registers D13 through D16) is written into the output register(s) being pointed at by registers D5 and D6 (*STEP #1 ACTIVE).*

When the time specified by register D11 has accumulated in register D3, the output pattern associated with Sequencer step #2 (registers D19 through D22) are written into the output registers being pointed at by registers D5 and D6 (*STEP #2 ACTIVE).*

Register D3 resets and begins timing again. When the time specified by register D17 has accumulated in D3, the advance to step #3 occurs (*STEP #3 ACTIVE).*

This process continues as long as bit 1 of register D2 is set to "1" and bit 5 of register D2 (AUTO/MANUAL select) is set to "0". When the last step in the Sequencer has been executed, bit 6 of register D2 (REPEAT/SINGLE PASS select) is checked to determine whether the Sequencer should "wrap around" to its step #1 or halt. If the Sequencer halts (bit 6 = "0"), all outputs are turned OFF.

POSSIBLE MODIFICATIONS: While the Time-Advanced Sequencer is running, the following modifications can be made:

- Instead of incrementing *forward,* the Sequencer can be set to operate in *reverse* (set bit 2 of D2 to "1").
- Upon step completion, *non-sequential* operation can be obtained (Bits 8 and 9 in register D2: bit 8 set to "1" executes the next step according to register D7, while bit 9 set to "1" non sequentially executes the next step in accordance to the contents in register D8). *Instantaneous* non-sequential operation can also be obtained by altering register D1.
- The *Current Step Running Time* can be held (inhibited from timing) by setting bit 3 of register D2 to "1".
- If the same output pattern must be maintained for more than the maximum time value (3,276 seconds), simply repeat the output pattern in multiple steps with appropriate Time Advance conditions.
- A *step advance inhibit* is enabled by setting bit 4 of register D2 to "1". This inhibits an advance even if the time requirement for the advance has elapsed.

CAUTION
(For Rev. 1 Firmware ONLY)

If a step advance is inhibited by the user from occurring when the Time Advance condition is satisfied, the Current Step Running Time register D3 continues to time. If any single step duration could exceed 3,276 seconds, a rung similar to the following should be programmed to prevent a time mis-compare that could result in a failure to advance when enabled:

[IF SXXXX ≥ 32,760]---[LET SXXXX = 32,760]

where XXXX is the register address represented by D3.

INPUT SEQUENCING ONLY

The following parameters apply to an input-advanced Sequencer:

- Operation is *automatic;* bit 5 of register D2 = "0".
- Load register D4 with the address of the input register being pointed at. This is used for comparison to determine advance conditions.

NOTE: *Register D4 is simply a pointer and is not the register that is actually compared.*

- All Time Condition advance registers (D11, D17, etc., depending on the number of steps being used) should be set to "0".
- Input Condition advance registers (D12, D18, etc., depending on the number of steps) should be preset with the bit pattern which, when compared to the register that's pointed at by D4, allows a step advance.

NOTE: *The bit pattern of "1"s and "0"s does not have to match exactly; it is only the "1"s in the Input Condition Advance register which have to be matched by the corresponding bits in the pointed to register. The "0"s are not matched. This means that if an advance is dependent upon one or more bits being "0", these bits need to be inverted prior to being compared.*

EXAMPLE: If the necessary condition for advancing to the next step is having bits 2 and 3 ON, then these two bits should be the only bits set in the Input Condition advance register. As soon as bits 2 and 3 are ON in the pointed-to register, a step advance is enabled *regardless* of how many other bits are ON in the register being pointed at.

OPERATION: When bit 1 of register D2 is set to "1", the output pattern associated with Sequencer step #1 (registers D13 through D16) is written into the output register(s) pointed at by registers D5 and D6 (*STEP #1 ACTIVE*).

When the bit pattern of "1"s specified in register D12 is matched by the pattern in the input register being pointed at by D4, the following occurs: The output pattern associated with Sequencer step #2 (registers D19 through D22) is written into the output register(s) being pointed at by D5 and D6 (*STEP #2 ACTIVE*).

Comparison for the third step advance is made with register D18. This process continues for as long as bit 1 of register D2 is set to "1" and bit 5 of register D2 (AUTO/MANUAL select) is set to "0".

After the last Sequencer step is executed, bit 6 of register D2 (REPEAT/SINGLE-PASS select) is checked to determine whether the Sequencer should "wrap around" to step #1 or halt. If the Sequencer halts (bit 6 = "0"), all outputs are turned OFF.

POSSIBLE MODIFICATIONS: While the Input Advanced Sequencer is running, the same modifications apply as for the Time Advanced Sequencer (Forward/Reverse stepping, non-sequential stepping, etc.). One exception is the *current step running time* parameter, which is not applicable in a Sequencer based solely on input conditions.

COMBINATION TIME AND EVENT/INPUT SEQUENCING

The following parameters apply to a Sequencer whose advances are based on *both* time and inputs:

- Operation is *automatic;* bit 5 of register D2 ="0".
- Register D4 should point to the register address whose bit pattern is used to compare for a step advance.

  NOTE: *Register D4 is simply a pointer and is not the register that is actually compared.*

- The Time Condition advance registers (D11, D17, etc., depending on the number of steps) should be loaded with times in tenth-second (0.1 second) increments.

  NOTE: *If a given step advance is to be based solely on input, "0" should be entered into the Time Condition register for that step.*

- The Input Condition advance registers (D12, D18, etc., depending on the number of steps) should be loaded with the bit patterns which, when compared to the register pointed at by D4, allow a step advance. As explained in the *INPUT SEQUENCING* section, it is only the "1"s in the Input-Condition advance register that are compared.

  NOTE: *If a given step advance is based solely on time conditions, "0" should be entered into the Input Condition register for that step.*

- If *both* time and event sequencing are specified in a step, an advance is not enabled *until the input condition has been met after the time condition is satisfied* (unless bit 10 of register D2="1"). This occurs even if the input condition has previously been met. If bit 10 of register D2 = "1", then a step advance is enabled upon *either* condition being satisfied.

OPERATION: When bit 1 of register D2 is set to "1", the output pattern associated with Sequencer step #1 (registers D13 through D16) is written into the output registers being pointed at by registers D5 and D6 (*STEP #1 ACTIVE).*

When register D3 is greater than or equal to *D11* (i.e., the time condition is satisfied), the pattern of "1"s specified by register D12 is compared to the input register pointed at by register D4. When all the "1"s in D12 are matched by the register pointed at by D4, the output pattern associated with Sequencer step #2 (registers D19 through D22) is written into the output register(s) being pointed at by D5 and D6 (*STEP #2 ACTIVE).*

CAUTION
(For Rev. 1 Firmware ONLY)

***If a step advance is inhibited by the user* from occurring when the Time Advance condition is satisfied, the Current Step Running Time register D3 continues to time. If any single step duration could exceed 3,276 seconds, a rung similar to the following should be programmed to prevent a time mis-compare that could result in a failure to advance when enabled:**

[IF SXXXX ≥ 32,760]---[LET SXXXX = 32,760]

where XXXX is the register address represented by D3.

Register D3 resets and begins timing again. Comparison for the third step advance is made using registers D17 (for time) and D18 (for input). This process continues as long as bit 1 of register D2 is set to "1" and bit 5 of register D2 (AUTO/MANUAL select) is reset to "0".

After the last Sequencer step is executed, bit 6 of register D2 (REPEAT/SINGLE-PASS select) is checked to determine whether the Sequencer should "wrap around" to step #1 or halt. If the Sequencer halts (bit 6 = "0"), all outputs are turned OFF.

POSSIBLE MODIFICATIONS: While the combination Time and Input Advanced Sequencer is running, the same modifications apply as for the Time Advanced Sequencer (Forward/Reverse stepping, non-sequential stepping, etc.). One exception is the *current step running time* parameter, which is not used if a particular Sequencer step is based solely on input conditions.

MANUAL SEQUENCING

A manual Sequencer does not rely on time and/or input conditions to advance. Instead, advance is determined solely by the state of bit 7 in register D2, which is toggled manually.

NOTE: *If an input condition exists for advance, it is ignored. If a time condition exists, register D3 do not accumulate time. If desired, the input and/or time conditions for advance can be loaded in anticipation of automatic operation, because all input and time conditions are simply ignored in the manual Sequencer,*

The following parameters apply to a manual Sequencer:

- Operation is *manual;* bit 5 of register D2 is set to "1".
- The Sequencer is advanced by toggling bit 7 of register D2 from "0" to "1".

OPERATION: When bit 1 of register D2 is set to "1", the output pattern associated with Sequencer step #1 (registers D13 through D16) is written into the output registers being pointed at by D5 and D6 (*STEP #1 ACTIVE).*

If bit 5 of register D2 is set to "1", the next step advance occurs when bit 7 of register D2 is toggled from "0" to "1". At this time the output pattern associated with Sequencer step #2 (registers D19 through D22) is written to the output register(s) pointed at by D5 and D6.

NOTE: *To prepare for another step advance, bit 7 of register D2 must be reset to "0" before being toggled back to "1".*

Sequencer behavior can be set to automatic at any time by resetting bit 5 of register D2 to "0". After the last Sequencer step is executed, bit 6 of register D2 (REPEAT/SINGLE-PASS select) is checked to determine whether the Sequencer should "wrap around" to step #1 or halt. If the Sequencer halts (bit 6 = "0"), all outputs are turned OFF.

POSSIBLE MODIFICATIONS: While the manual Sequencer is running, the same modifications apply as for the automatic advance Sequencer (Forward/Reverse stepping, non-sequential stepping, etc.). Bit 4 of register D2 only applies to the AUTO mode operation and *cannot* be used to inhibit a MANUAL step advance.

NON-SEQUENTIAL STEPPING

Regardless of whether the Sequencer is operating in the manual or automatic mode, it is possible to alter the default incremental-forward stepping mode in the following ways:

1. Reverse advance (decrement step).
2. Non-sequential advance (i.e., jump to a step not immediately preceding or following the current step).
3. Immediate and unconditional non-sequential advance to another step, even though the advance conditions for the present step are not satisfied.
4. Inhibit a step advance while in the AUTO mode, causing the current step to be maintained despite any satisfied conditions for automatic advance that are present.

The non-default advance conditions are as follows:

Reverse (Decremental) Stepping. Set bit 2 of register D2 to "1", causing the Sequencer to step backward (reverse mode). Decrementing from Sequencer step #1 causes the Sequencer to go to step 0 (all outputs OFF) unless bit 6 of register D2 is set (REPEAT mode), in which case the Sequencer goes to the last step.

NOTE: *Bit 2 of register D2 has a lower priority than bits 4 (Step Inhibit), 8 or 9 (Non-Sequential Step) or setting Register D1 to the desired step.*

If the reverse mode is selected prior to Sequencer enabling, the Sequencer - upon initialization - jumps to the last step (except Rev. 1 Model 400 firmware. With Rev. 1, the jump is to the first step).

Non-Sequential Advance. Set bit 8 of register D2 to "1", and load register D7 (first non-sequential step register) with the number of the next desired step, or set bit 9 of register D2 to "1" and load register D8 with the number of the next desired step. Bit 8/Register D7 and Bit 9/Register D8 function identically; two pairs are provided for user convenience.

NOTE: *If <u>both</u> register pairs are enabled simultaneously, Bit 8/Register D7 takes precedence.*

When either of these bit/register pairs is activated, the Sequencer goes to the step specified in either register D7 or D8 when the next step advance occurs. The number entered in these registers must be a valid step number (zero is NOT a valid step number); otherwise no advance takes place. Outputs remain as defined for the current step and bit 18 of register D2 is set to "1" to indicate an error.

If a non-sequential advance is specified prior to Sequencer enabling, the Sequencer advances - upon initialization - to the specified non-sequential step.

Immediate And Unconditional Non-Sequential Advance. Loading a step number into register D1 (Current Step Number) causes the Sequencer to jump unconditionally to that step upon the next program scan. The Sequencer detects this by storing the step executed during the previous scan in register D9.

At the beginning of each new scan, D9 is compared with D1. If they differ, the Sequencer immediately jumps to the new step specified in D1. Normal Sequencer operation resumes from that point – register D3 resets and begins timing, and the Input Condition advance register for the new step is used for comparison.

NOTE: *The new step that is jumped to must be valid for the Sequencer range defined. If the step is invalid, no step change occurs; outputs remain as defined for the previous step and bit 18 of register D2 is set to "1" to indicate the error.*

If an immediate and unconditional non-sequential advance is specified prior to Sequencer enabling, the Sequencer - upon initialization - advances to the first step for one scan before jumping to the selected step. Similarly, if a step advance is enabled while an immediate and unconditional non-sequential advance is specified, the advance goes active for one scan before reverting to the specified step.

Inhibit a Step Advance (AUTO mode only). Set bit 4 of register D2 to "1". This prevents the Sequencer from advancing beyond the current step, even if conditions for advance are satisfied when in the automatic mode.

CAUTION

(For Rev. 1 Firmware ONLY)

If a step advance is inhibited by the user from occurring when the Time Advance condition is satisfied, the Current Step Running Time register D3 continues to time. If any single step duration could exceed 3,276 seconds, a rung similar to the following should be programmed to prevent a time mis-compare that could result in a failure to advance when enabled:

[IF SXXXX ≥ 32,760]---[LET SXXXX = 32,760]

where XXXX is the register address represented by D3.

The Automatic Step Advance Inhibit applies regardless of whether the step change is to be forward or reverse.

As soon as bit 4 of register D2 is reset to "0", output control reverts to control of the automatic advance condition. For example, if the only condition for advance is inputs (which happen to be satisfied), the step advance occurs the instant that bit 4 is reset to "0".

If a step advance is inhibited *prior* to Sequencer enabling, the Sequencer - upon initialization - advances to the *first* step. The *next* step advance is inhibited, provided the inhibit remains selected and the Sequencer is in AUTO mode.

Bit 4 only applies to automatic mode operation (bit 5 of register D2 = "0") and does not inhibit a manual or Immediate And Unconditional Non-sequential Step Advance.

11.4 Application Considerations

This section presents a collection of miscellaneous considerations for Sequencer operation.

11.4.1 ERROR CONDITIONS

Bit 18 of the Operating Mode register (register D2) is used to flag invalid Sequencer operating conditions. When an invalid operation is detected by the Sequencer, bit 18 is set to "1".

If the error is generated while the Sequencer is running, the condition of the previous step is retained. If the error exists prior to Sequencer enabling, no step execution occurs until the error is corrected.

The most common reasons for bit 18 to be set are listed below:

1. An undefined (non-existent) step has been specified in registers D1, D7 or D8.

   NOTE: *D7 or D8 are not checked until either bit 8 or bit 9 of the Operating Mode register = "1".*

2. The number of steps specified exceeds 255 (the Sequencer does not commence operation at all until this error is corrected).

3. An invalid Register Block starting address has been given in the Sequencer's configuration rung (Figure 11.1).

4. An invalid Output Width Definition has been entered into register D6 (width must be in the range 1-64).

5. Register D4 = "0", but a Step Input Advance condition register (D12, D18, etc.) exists and is not equal to "0".

When any of these error conditions appear, the Sequencer is reporting that it has been asked to perform an illegal operation.

When defining a register block for the Sequencer to use, make sure that sufficient registers exist between the start of the block and the last available register in the Model 400 (register 4000 is the last). Otherwise, the Sequencer flags an invalid operating condition and sets bit 18 of the register indicated to the left of the " = " sign.

To guarantee sufficient registers, apply the following equation (from Section 11.2):

Total Registers Per Sequencer
= [6x (# of steps) + 10]

Subtract the total registers per Sequencer from 4000 to obtain the *highest* address that the Sequencer can be started at.

EXAMPLE: A 50-step Sequencer is to be programmed into the Model 400. From the equation above, the registers required for this Sequencer are:

$$[6 \times (50) + 10] = \mathbf{310}$$

The highest address at which this Sequencer can be located is therefore:

$$4000 - 310 = \textbf{register \#3690}$$

The Sequencer's register block must begin at an address no higher than 3690.

NOTE: *Do not use registers allocated for a Sequencer for any other purpose.*

11.4.2 OPERATING MODE REGISTER STATUS BITS

Bits 17-23 of the Operating Mode register (D2) are read-only bits that are used to monitor the status of the Sequencer. The addresses of these bits can be programmed as contacts in the Model 400's user memory to provide detailed information on Sequencer operation. All of these status bits are reset to "0" when the Sequencer is reset (when bit 1 of register D2 is set to "0"). Each of the status bits is described below:

Bit 17 - Sequence Complete. When set, indicates that the Sequencer has just completed the final step. If bit 6 of register D2="0" (SINGLE-PASS SEQUENCE selected), bit 17 remains set at "1" until the Sequencer is reinitialized by toggling bit 1 of register D2 from "0" to "1". If bit 6 of register D2 = "1" (REPEAT SEQUENCE selected), bit 17 is set at "1" for only a single Model 400 scan as the *Sequencer "wraps around" from the final step to the* first step.

Bit 18 - Error Bit. This bit is described in Section 11.4.1.

Bit 19 - Manual Advance History. When set to "1", a manual advance has occurred. This bit is used to insure that the manual advance bit (bit 7 of register D2) is transition-sensitive.

Bit 20 - Time Advance Enabled. When set to "1", the time advance condition for the current step has been satisfied.

Bit 21 - Input Advance Enabled. When set to "1", the input advance condition for the current step has been satisfied.

Bit 22 - Reset/Enable History. When set to "1", the Sequencer has been enabled. This bit is used to ensure that the reset/enable bit (bit 1 of register D2) is transition-sensitive.

Bit 23 - Step Advance Flag. When set to "1", a step advance has just occurred. Typically this bit is set for a single scan.

11.4.3 OTHER CONSIDERATIONS

Although *all* possible operating configurations cannot be examined, the following items should be considered when programming the Sequencer:

- OUTPUT STATES. Output states associated with a given step are written only if a step change occurs and are not refreshed each time the Sequencer is scanned. If the CPU is halted or power is removed when the Sequencer is in the middle of executing a step, the output states are not updated when programming execution resumes **until** a step change actually occurs. When the CPU resumes operation from a halt condition or power cycle, the external output registers are automatically reset. Sequencer output states that are directly written to external outputs remain OFF until the currently executing step has finished and a step advance has occurred. **If the user wishes to preserve the Sequencer output conditions for a given state following a processor halt or power cycle,** specify an internal storage register(s) in D5 and D6 and perform a data transfer from the storage register to the external output register.

- EXTERNAL OUTPUTS. Due to Image Table updating, output states that are associated with a given step are not written to external outputs *until the end of the Model 400's ladder scan.* The only exception to this is if the Immediate I/O Update function (bit 7 of 8176) is used.

- RUNG PROGRAMMING. It's recommended that any rungs which are used to modify the Sequencer's operation be programmed *prior* to the Sequencer initiation rung.

- SUBROUTINES. Since Sequencers are designed to be executed during each ladder scan, they should NOT be used in subroutines or in any part of a ladder diagram that could be skipped. Also, no logic should be placed in front of the SPEC 65 rung, to ensure the instruction is always executed.

- MCR. When preceded by a disabled MCR, the Sequencer resets. This is because the RESET/ ENABLE bit (programmed as a coil) is OFF.

11.5 Examples

This section provides a simulated Sequencer to help clarify the Sequencer's set-up and operating procedures.

EXAMPLE ONE SET-UP:

Assume the following Sequencer conditions for the example:

Number of Steps	20
Registers Required	130 = [(6×20) + 10]
Beginning Register Address	0751
Ending Register Address	0880
Output Register Address	0004
Output Width	12 bits (register #4, bits 1 through 12)
Input Register Address	0001 (any or all of the 16 bits can be used to establish advance conditions).

Upon Sequencer initialization (step #1), assume that bits 2, 3 and 4 of the output register (#4) are to be set to "1". The initiation rungs needed are shown in Figure 11.7.

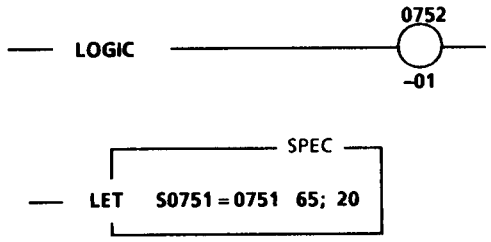


Figure 11.7 Example One Initiation Rungs

In the first rung shown in Figure 11.7, **0752-01** is the RESET/ENABLE bit for the Sequencer. In the second rung, **0751** is the address that identifies the start of the register block. **SPEC 65** identifies the special instruction as a Sequencer, while **20** is the number of steps in the Sequencer.

In accordance with Section 11.3.1, the following registers have to be set up for the first step (step #1) in Example One:

0752 (D2) .. Operating Mode (assume for now the default operation).

0754 (D4) .. Pointer to Input Register 0001. Preset to 1.

0755 (D5) .. Pointer to Output Register 0004. Preset to 4.

0756 (D6) .. Output Width Definition. Preset to 12.

0761 (D11) Time Condition Advance. Assume that bits 2 through 4 of register 0004 must be ON for 60 seconds to satisfy the time advance requirement. Preset to 600 (60 ÷ 0.1 sec. increments).

0762 (D12) . Input Condition Advance. Assume a step advance is enabled when bits 1, 3 and 4 of register 0001 are set to "1". Preset to 13 (decimal value. For corresponding binary pattern, see Figure 11.10).

0763 (D13) . Step #1 Output States bits 1 through 16. Since bits 2, 3 and 4 of register 0004 is set to "1", preset to 14 (decimal value). For the corresponding binary pattern see Figure 11.8.

0764 through 0766 (D14 through D16) ..... Step #1 Output States bits 17 through 64. Since the Sequencer's output width is less than 17, these registers are not applicable.

The remaining 19 steps in the Sequencer must be set up separately and are not shown here.

EXAMPLE OPERATION

Before bit 1 of register 0752 (D2) is energized, registers 0751 and 0753 equals zero and bits 1 through 12 of register 0004 is reset to "0" every time the initiation rungs (Figure 11.7) are scanned by the *Model 400.*

When bit 1 of register 0752 is set to "1", the following occurs:

1. The Sequencer sets register 0751 to "1".

2. (Also see "2A" below). Output information in *registers 0763 through 0766 is transferred to the* output registers specified by 0755 (D5) and 0756 (D6). Since D5 = 4 and D6 = 12, bits 1 through 12 of register 0763 are transferred to bits 1 through 12 of register 0004. Since register 0763 = 14, bits 2 through 4 of this register are set to "1" while bit 1 and bits 5 through 12 are set to "0" as shown in Figure 11.8 below:

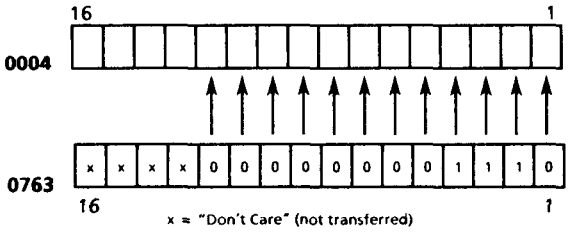


Figure 11.8 Transfer Of 12-Bit Output Width

2A. If the output width in this example was specified as *52* instead of 12 (register 0756 = 52), *additional output registers would be used.* As shown in Figure 11.10, all 16 bits of registers 0763, 0764 and 0765, along with bits 1 through 4 of register 0766 are written to registers 0004, 0005, 0006 and the first four bits of 0007. If the bit patterns of Figure 11.10 are involved in a transfer, it is necessary to preset register 0764 with the decimal value -24,562 (A00EH), *register 0765 with decimal 16,391 (4007H), and* register 0766 with decimal 15 (000FH). These settings allow the multi-register transfer to take place.

3. Bit 23 of register 0752 is set to "1" for one scan.

4. Register 0753 (D3) begins to time (to be *compared with register 0761) to determine* Time Advance enable.

5. The bit pattern of register 0001 (the Input Register pointed at by register 0754) is compared to register 0762 to see if the condition for Input Advance is satisfied. The Input *Advance does not enable a step advance until* the time condition (register 0753 = register 0761) has been satisfied.

6. Step advance to step #1 is now complete. Assuming no Operating Mode register bits are set, the next step advance occurs when bits 1, 3 and 4 (specified in register 0762) of register 0001 *(pointed at by register 0754) are set to "1" as* shown in Figure 11.9 below.

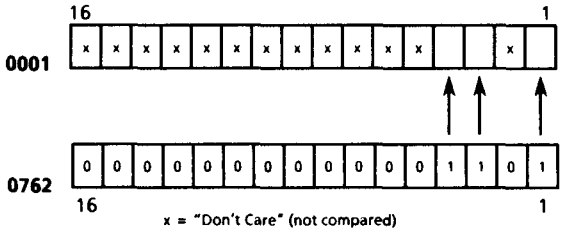


Figure 11.9 Input Advance Condition Satisfied

Before this comparison occurs, the time condition for advance (60 seconds) must be satisfied. If a TRUE comparison does not occur before 3,276 seconds has elapsed, program the following rung (not required for Rev. 2 or later):

---[IF S753 ≥ 32,760]---[LET S753 = 32,760]

NOTE: *If it is desired to have the step advance in this example based solely on time, presetting register 0762 to zero allows the advance to occur after 60 seconds. If a strictly input based advance is desired, presetting register 0761 to zero allows an advance as soon as bits 1, 3 and 4 of register 0001 are set to "1".*

The following is a list of options that are not necessary for basic Sequencer operation. The description of Sequencer operation continues (at step #7) after the options.

OPTIONS:

Now that the Sequencer is enabled, a variety of operating modes can be selected by altering the bits in the Operating Mode register (register 0752). Some possibilities are listed below:

Reverse Sequencing- Set bit 2 of register 0752 to "1", and the Sequencer runs in reverse. If the Sequencer is currently at step #1, enabling bit 2 advances the Sequencer to step #20 (the last step) instead of to step #2.

Prevent Time Accumulation - Set bit 3 of register 0752 to "1" and no time accumulates in the Current Step Running Time register (0753).

Inhibit A Step Advance - Set bit 4 of register 0752 to "1" and the Sequencer does not advance from the current step – even if time (register 753 = 600) and/or input (bits 1, 3 and 4 of register 0001 = "1") conditions are satisfied. Also, output register 0004 does not change.

Manual Operation - Set bit 5 of register 0752 to "1" and the Sequencer ignores both time and input advance conditions. Toggle bit 7 of register 0752 to manually step the Sequencer.

Non-Sequential Stepping - Bits 8 and 9 of register 0752 are used to make the Sequencer jump to a step that is not in numerical order (for example, jump from step #1 to step #4, ignoring #2 and #3). Register 0757 or 0758 needs to be loaded with a valid step number (1 to 20), otherwise the current step is maintained. If bit 8 or 9 of 0752 is set, and advance conditions are satisfied, the Sequencer jumps to the step specified in register 0757 (if bit 8 is set) or register 0758 (if bit 9 is set).

NOTE: *If an immediate jump to a non-sequential step is desired, load the valid step number (1-20) into register 0751. Step advance occurs even if time and/or input conditions are not satisfied.*

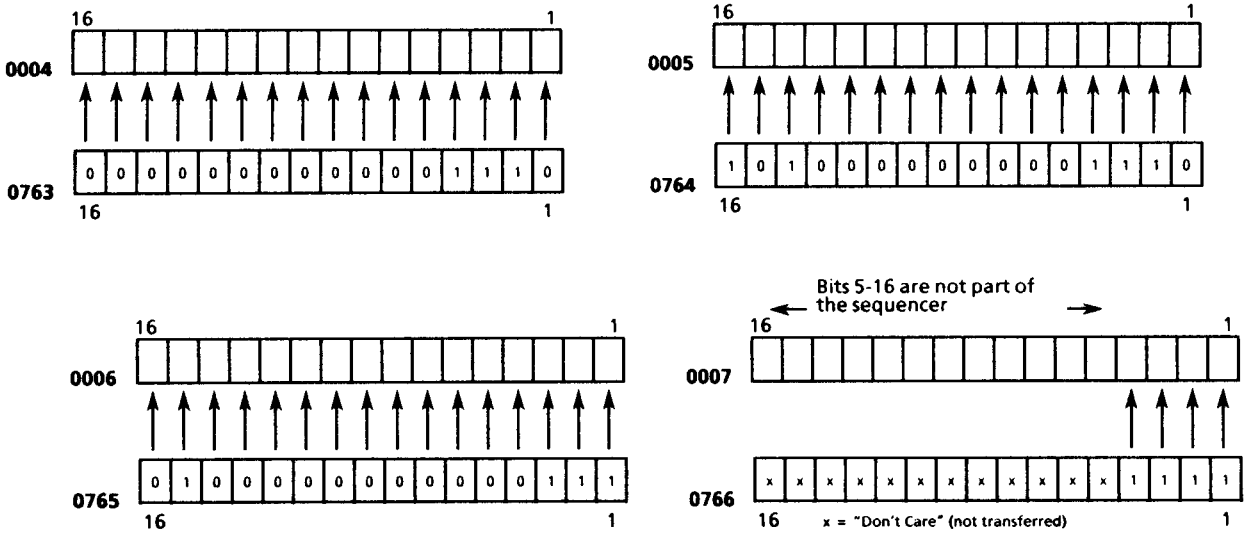


Figure 11.10 Transfer Of 52-Bit Output Width

Time or Input Condition Advance - Set bit 10 of register 0752 to "1" and a step advance is enabled as soon as *either* register 0753 = 600 (60 seconds) *or* bits 1, 3 and 4 of register 0001 are set to "1".

7. Sequencer step #2. Assume the initial conditions of this example are unchanged and that no Operating Mode register bits are set to "1". Fix the output bit pattern for step #2 such that bits 1, 4 and 5 of register 0004 are to be set to "1" (all other bits "0"). This is accomplished by setting register 0769 to 25 (0019H). Figures 11.11A and 11.11B are the requirements and results of Sequencer step #2.

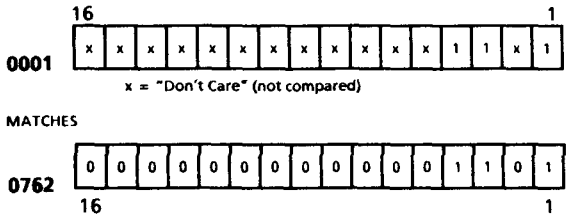


Figure 11.11A Step #2 Advance

When register 0001 matches register 0762 (as shown in Figure 11.11A), and register 0753 ≥ register 7610 (step #1 has been running for at least 60 seconds), the following occurs:

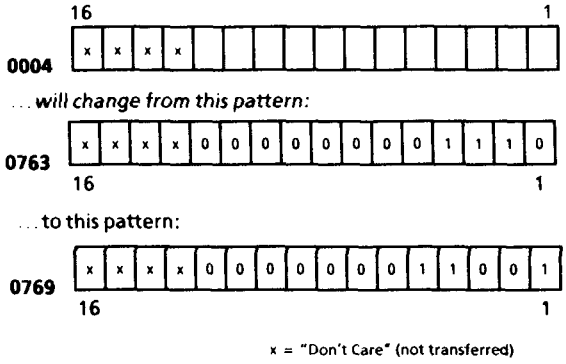


Figure 11.11B Step #2 Result

For *manual* operation (bit 5 of register 0752 = "1"), register 0004's pattern would change from the pattern in register 0763 to the pattern of 0769 when bit 7 of 0752 is toggled from "0" to "1".

This procedure is followed for each step as long as the Sequencer is enabled. If the Sequencer is disabled by resetting bit 1 of register 0752, the outputs are turned OFF and the current step register (0751) is reset to 0. Upon reaching the last step (step 20), bit 6 of register 0752 is monitored. If bit 6 is set, the next step advance is to step 1 and the Sequencer operation continues. If bit 6 is reset, the next step advance concludes the sequencing operation and all outputs are turned OFF.

12 BATTERY BACKUP AND CLOCK/CALENDAR

12.1 Backup Battery

12.1.1 DESCRIPTION

The Model 400 uses two methods to supply backup power to the onboard volatile RAM and real-time clock/calendar circuitry in the event that power to the power supply is interrupted:

1. Batteries located in the SY/MAX power supply (PS-21, 31, etc) are the primary source of backup power for the Model 400 RAM and real-time clock/calendar in the event that +5VDC from the power supply is lost. Refer to Instruction Bulletin 30598-156-XX or 30598-159-XX for information on the power supply batteries.

2. The Model 400's onboard lithium battery supplies power to the RAM and real-time clock/calendar if the voltage from the batteries in the power supply is not sufficient to retain valid data in the RAM. The battery also maintains the RAM contents and clock/calendar accuracy if the processor is removed from the rack or shipped for use in another location.

12.1.2 LOW BATTERY INDICATION

The condition of either the SY/MAX power supply batteries or the Model 400's onboard lithium battery is indicated with the BATTERY LOW LED on the front panel of the processor. See Section 4.2.6 of this manual for a complete explanation of this LED.

NOTE: *If the onboard battery is not installed, the BATTERY LOW indicator signals a low internal battery condition.*

12.1.3 BATTERY LIFE EXPECTANCY

The life expectancy of the onboard lithium battery is dependent on the time that the battery must supply current to the Model 400, and on the physical age of the battery (see Section 2.1 for specifications).

Figure 12.1 gives a rough estimate of how long the Model 400's RAM is supported when a battery ranging from new to five years old is installed in a Model 400. The BATTERY AGE assumes the battery has been installed in the Model 400 but *not supplying current*, and that the Model 400 has been at its maximum operating temperature of 60°C. The RAM DATA RETENTION TIME is based on a Model 400 at 40°C, since the processor is not at operating temperatures while on lithium battery backup.

12.1.4 REPLACEMENT PROCEDURE

The internal lithium battery is replaced by unscrewing the small access cover on the front panel of the Model 400. The battery can be changed while the Model 400 is operating. If the processor is installed in a rack which is supplying either regular or battery power, there is an unlimited time in which to replace the battery.

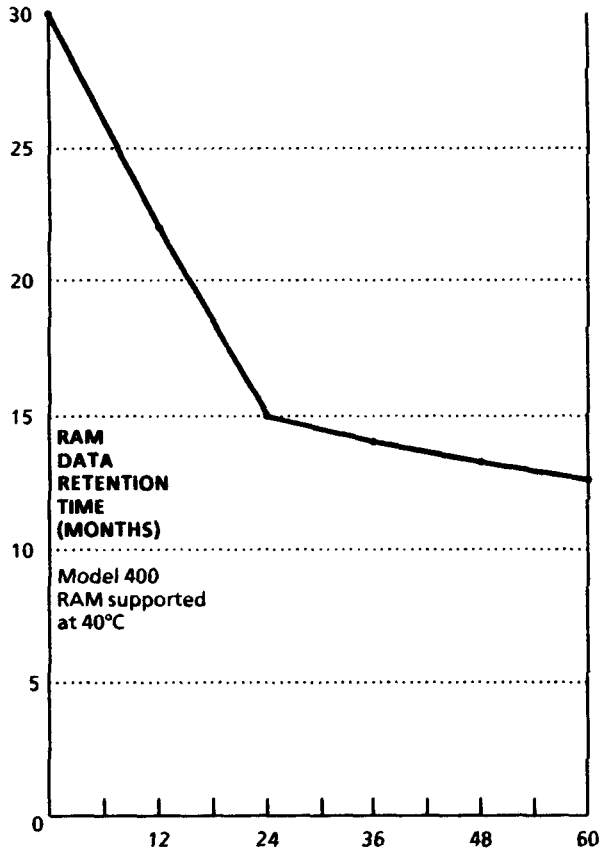


Figure 12.1 Backup Battery Life

NOTE: *For maximum onboard battery life, the Model 400 should remain in the rack, connected to the power supply, as much as possible. In this way the power supply batteries – not the internal battery – are supplying any needed backup power.*

12.1.5 BATTERY SPECIFICATIONS

Square D Part Number	29904-08961
Manufacturer	Tadiran Electronic Industries Inc.
Mfg. Type Number	TL-5104
Mfg. Catalog Number	15-51-04-210-000
Rated Voltage	Lithium AA, 3.5 to 3.6 V
Capacity (200 uA @ 3V) ...	1.9 Amp Hours (Typical cells stored at 25°C for one year)

(alternate manufacturer source is SAFT, battery type LS6, catalog number 500200)

12.2 Real-Time Clock/Calendar

The Model 400 is equipped with a battery-backed real-time clock/calendar. The clock/calendar keeps track of seconds, minutes, hours, day of the week, day of the month, month, and year and clock compensates for leap years.

Values for the real-time clock/calendar reside in registers 8109 through 8115. These registers are updated in the Model 400 image table each hour by the regulation circuitry within the real-time clock. In between the hourly updates, the clock is updated second-by-second in the software.

12.2.1 SPECIFICATIONS

Format	24-hour("military") time. If 12-hour (AM/PM) mode is desired, user program must perform the conversion.
Accuracy	Less than one second per day @ 25°C, 16 seconds per day over the -32° to 140°F (0° to 60°C) range.
Set Method	Write to registers using one of the following methods: • Model 400 user program • Programming equipment • Other processors • D-LOG Module

12.2.2 LOADING THE TIME/DATE

Model 400 registers 8108 through 8115 are assigned to the clock/calendar. Figure 12.2 shows the function of each of these registers:

REGISTER	FUNCTION	DECIMAL RANGE
8108	Set clock (see Section 12.2.3)	-1, 0, or +1
8109	Seconds	0–59
8110	Minutes	0–59
8111	Hours	0–23
8112	Day of week	1–7 (1 = Sunday)
8113	Day of month	1–31
8114	Month	1–12
8115	Year	1980 - 2079

Figure 12.2 Clock/Calendar Registers

Set the current time of day in the clock by the following procedure:

1. Using a CRT Programmer or equivalent, load "-1" into register 8108. This interrupts the updating of the image table clock registers; it does not affect the real-time clock itself.

2. Access each of the registers shown in Figure 12.2 by *starting with register 8109*. Register 8108 is only used to transfer data to the clock.

3. While in DATA ENTER mode, load each of the registers 8109-8115 with the appropriate value. Since these are *control* registers, it is necessary to strike the ENTER key multiple times to load the values.

EXAMPLE: To set the current time as 3:10 PM on Friday, February 26th, 1988, the following values would be entered into *registers 8109--8115:*

8109 00

8110 10

8111 15

8112 6

8113 26

8114 2

8115 1988

12.2.3 *SETTING THE CLOCK*

After the loading procedure in Section 12.2.2 has been completed, the values in the image table must be transferred to the clock. This is accomplished by loading the decimal value **0** into register 8108. The *instant that this value is loaded into register* 8108, the clock resumes keeping time using the loaded values.

TO UPDATE AND RESTART THE CLOCK:

8108 0

12.2.4 CLOCK MANIPULATION

NORMAL OPERATION (Register 8108 set to 0)

When manipulating the clock, it's important to realize that the clock circuitry itself is running regardless of whether the Model 400 is powered up or not.

During "normal" clock operation, the user-accessed image table registers are updated directly from the clock once each hour. During each hour, the Model 400's oscillator regulates the updating of the image table each second. The specified drift of the software oscillator while the Model 400 is powered up is ± 1.8 seconds per hour. This drift is eliminated each time the real-time clock performs its hourly updates on the image table registers.

SET CLOCK OPERATION (Register 8108 set to -1)

To set the clock either via DATA ENTER mode or an instruction in the user program, it is first necessary to set register 8108 to "-1". This prevents the image table from being updated by *either* the real-time clock or by the Model 400's software oscillator.

NOTE: *The clock itself continues to run internally when register 8108 is set to "-1".*

After the desired values have been loaded into registers 8109-8115 (see Section 12.2.2), reset register 8108 to "0". This action transfers the values in 8109-8115 to the real-time clock itself and normal operation resumes (i.e., the image table registers are updated each second by the software oscillator and each hour by the real-time clock).

NOTE: *If out-of-range values are entered into any register, the values are rejected and an ERROR 42 is displayed on the programmer.*

STOPWATCH OPERATION (Register 8108 set to + 1)

A special operational mode is enabled when register 8108 is set to "+1". In this "stopwatch" mode, the hourly updates of the image table by the real-time clock are inhibited, but the second-by-second updates by the software oscillator continue. When register 8108 is reset to "0", normal clock operation resumes and the image table registers are updated with the most current hourly time in the real-time clock.

This stopwatch mode can be very useful for timing discrete events. Since the software oscillator continues to run, time can be accumulated without being overwritten by an update from the real-time clock. At the conclusion of the timed event, resetting register 8108 to "0" resumes normal operation and registers 8109 to 8115 are updated automatically by the real-time clock. There is no need to reset the time of day since this is retained in the real-time clock during the event's timing.

The three modes of operation are summarized below in Figure 12.3.

REGISTER 8108 Set To:	Clock Mode	External Clock Registers Being Updated By:	
		SOFTWARE	REAL-TIME CLOCK
-1	Set Clock	NO	NO
0	Normal	YES	YES
+1	Stopwatch	YES	NO

Figure 12.3 Clock Operation Modes

In summary, when register 8108 is set to "-1" external clock registers are not updated by either the real-time clock or the software oscillator. This allows a user to preset the clock registers for the instant when register 8108 is user-reset to "0". The preset values are simultaneously transferred to the real-time clock at this point. If register 8108 is set to "+1", the clock registers operate in the "stopwatch" mode.

13 CONTROL REGISTERS

13.1 Control Register Overview

The Model 400 processor uses a total of 4096 registers. These processor registers are separated into two main groups referred to as *control registers* and *data registers*.

Data registers are used to represent external I/O, internal relay equivalents, or data storage.

Control registers are reserved for Model 400 control functions. Register addresses 8097-8192 are reserved for these control functions.

Either type of register consists of 32 bits. Bits 1 through 16 are referred to as the *data field*, while bits 17 through 32 are referred to as the *status field*. In most cases, the status field is either unalterable or unused. The structure of a data register is shown in Figure 13.1.

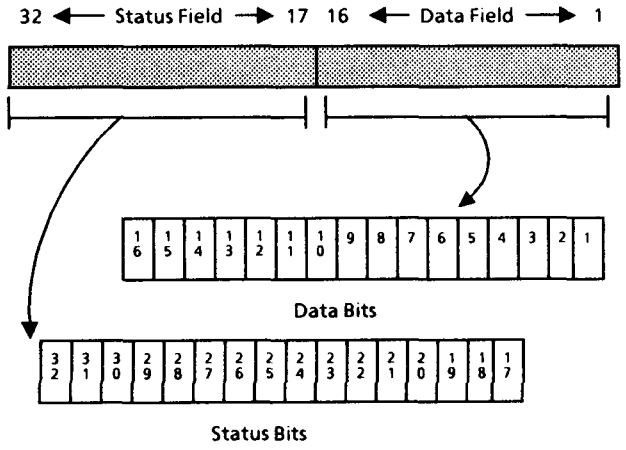


Figure 13.1 Register Structure

The control register addresses can be divided into two groups as follows:

Group #1 consists of addresses 8097-8178. Both the *data field and status field of group #1 can be* examined by the user, however only the data field can be altered.

Group #2 consists of registers 8179-8192. This group is used by the processor for certain "bookkeeping" functions. Only the data field is accessible to the user and can only be examined, not altered.

The examination of control registers and their individual bits can be accomplished by viewing them on a programmer or by using them as contacts within the ladder program.

WARNING

Altering control register bits with a programming device may effect actual equipment under control of the programmable control system. Make sure that all effects of altering control bits are understood prior to their alteration.

If control register bits are to be altered, the DATA mode of a programmer (CRT or Hand Held) can be used or the bit can be programmed in the ladder as a coil.

13.2 Listing of Registers

8001–8005: Command identifiers (not actual storage registers) which identify the ASCII output format.

These registers are used as commands to set the format of the Model 400's output data when the Model 400 is executing a PRINT statement and DO NOT actually exist as usable storage registers for other purposes. Available formats include:

- ASCII in decimal, hexadecimal, or binary
- "Raw binary" data
- Repeated character strings

Refer to the programming device's instruction bulletin for programming information and Section 10.6 for application information.

8006: Acts as the command identifier (not an actual storage register) for the ASCII input format. See Section 10.1.

8007-8096: Not available.

8097: Acts as a pointer for the indirect ROUTE list using the PRGMR (channel 1) port. See Section 5.5.

8098: Same function as 8097, except controls COMM (channel 2) port.

8099: PRGMR port (channel 1) attributes. Refer to the following tables and Section 10 for bit explanations of this register.

8100: COMM (channel 2) attributes - same bit patterns as 8099.

8101-8104: Not used.

8099 Bits 1-4: Baud Rate:

BIT PATTERN	BAUD RATE
0000	75
0001	110
0010	300
0011	1200
0100	2400
0101	4800
0110	9600
0111	19200

8099 Bits 5-6: Word Size:

BIT PATTERN	WORD SIZE
00	8 bit
01	7 bit
10	6 bit
11	5 bit

8099 Bits 7-8: Parity:

BIT PATTERN	PARITY
00	Even
01	Odd
10	None
11	None

8099 Bits 9-10: Stop Bits:

BIT PATTERN	STOP BITS
00	1
01	1.5
10	2
11	2

8099 Bit 11: ASCII Output Control Function (see registers 8001-8005)

BIT PATTERN	FUNCTION
0	XON/XOFF Disabled
1	XON/XOFF Enabled

8099 Bit 12: ASCII Input Null (00) Characters

BIT PATTERN	FUNCTION
0	Null ignored
1	Null stored

8099 Bit 13: Transmit Periodic DLE/ENQ Stream (SY/MAX Protocol-Handshaking Characters)

BIT PATTERN	FUNCTION
0	Suppressed
1	Active

8099 Bit 14: Mask 8th Bit of ASCII Output Word (Reset Model 400's 8th ASCII output Bit to "0")

BIT PATTERN	FUNCTION
0	No mask
1	Mask 8th bit

8099 Bit 15: CTRL-C Flag

BIT PATTERN	FUNCTION
0	Not Received
1	Received

8099 Bit 16: Clear Queued ASCII Input Buffers

BIT PATTERN	FUNCTION
0	Inactive
1	Active

8105: Pointer to the first register for Immediate I/O Update instruction. See bit 7 of register 8176.

8106: Register block size for the Immediate I/O Update instruction. See bit 7 of register 8176.

8107: Pointer to the Error Register Stack (16 register block). Can be set by the user to point to the beginning of a block of 16 registers that preserve error codes which appear in register 8175. Reserve this 16-register block - which must begin on or before register 3985 - exclusively for use as the Error Register block.

In normal operation, error codes appearing in register 8175 are zeroed when the fault is cleared. Register 8107 provides a valuable troubleshooting aid by allowing the user to create a history of error codes.

As each new error is received, it is inserted into the first register position in the 16-register stack. Errors already in the stack are pushed down one position. When the stack is full, the oldest error is pushed out of the stack and is lost.

For example, if a particular error code occurs more than once consecutively (i.e., if ERROR 20001 enters the stack four consecutive times), it only appears once in the stack. Thus, multiple failures may only produce one error code in the stack if the failures are all caused by the same error code condition.

8108: Clock - Command Register.

8109: Clock - seconds (0-59).

8110: Clock - minutes (0-59).

8111: Clock - hours (0-23).

8112: Clock - day of week (1-7, 1 = Sunday).

8113: Clock - day of month (1-31).

8114: Clock - month (1-12).

8115: Clock - year (1980-2079).

8116-8118: Not used.

8119: Last register defined for channel 2 of the 15th LI module.

8120-8163: For use in communicating with Local Interface modules. Each 2-channel LI module utilizes up to 3 processor control registers for monitoring I/O and system control communications. Registers 8161-8163 of this group are reserved for the first LI module as explained in the following:

(8161) Control Register for channel 2 of the first Local Interface module, is used to monitor and control the conditions of the remote racks that contain the register modules and I/O connected to channel 2 of this local interface. The bit pattern of 8161 is as follows:

Bits 1-8: The HALT/RUN bit can be set to force the corresponding drop to shut down.

Bit 9: The FREEZE/RESET bit allows the user to choose if the channel should freeze (remain in last previous state) or reset (turn OFF) when a communication error exists. 1 = FREEZE, 0 = RESET.

Bit 10: XMIT FAULT TOLERANCE bit. When set to "1", allows up to 10 attempts to re-establish communication prior to generating an error. When this bit is reset to "0", only three attempts are allowed.

Bit 13: The FAILURE OVERRIDE bit allows the system to continue operation should communication to the remote I/O racks fail. 1 = Failure Override Mode 0 = Normal Operation.

Bit 15: AUTO RESTART bit. When set, automatically attempts to re-establish communication should an I/O fault occur on the channel. 1 = Auto Restart ON; 0 = Auto Restart OFF. Must be set in conjunction with bit 13 to be practical.

NOTE: *The remainder of the bits in register 8161's data field are unused.*

(8162) Channel 1 of the first Local Interface module. (This register performs the same function as register 8161 does except it monitors and controls Channel 1).

(8163) Error code control register for the first local interface. Refer to Appendix B for the error code table.

8164: A 2.5 ms "tick" register. Increments every 2.5 milliseconds regardless of the RUN/HALT status of the Model 400. All 32 bits of the register are used to keep track of accrued time. Maximum accrued time is 4 months. This register is reset upon its "rollover" or power up and can be preset by a programmer or from the ladder program. Whenever the data field of register 8164 is reset to zero, the status field also reverts to zero.

If the contents of register 8164 are used in an IF or LET rung, register 8164 should be "ORed" with zero before performing the data compare (IF) or transfer (LET). This effectively masks off the 16 bits of the status field, and prevents an overflow condition from occurring if any of these bits are non-zero.

EXAMPLE: LET S3995=S8164 executes as expected until register 8164 exceeds +32,767, at which time the contents of register 3995 stops changing. Bit 18 (overflow) of register 3995 is set to indicate the overflow condition. If register 3995 should remain "free-running", use the following rung:

LET S3995 = S8164 $\vee$ 0

After reaching +32,767, the contents of register 3995 changes to -32,768 then increments back to zero.

8165: Used to establish safeguard rung parameters (see Section 6.8). Also contains the number of "ticks" remaining before the next timed interrupt is due to be called.

8166: Programmable minimum scan-time register. A positive number entered into this register establishes the minimum CPU ladder program scan time, to a resolution of 2.5 milliseconds. The Model 400 executes the user program followed by a series of "idle" instructions until the minimum scan time has elapsed. The desired scan time can be entered in 2.5 ms increments that are rounded up. For example, entering a minimum scan time of 98, 99 or 100 ms results in a ladder program scan time of not less than 100 ms and not more than 102.5 ms. Any scan time entered is subject to being increased by the I/O update time, plus the 5 ms port servicing delay.

8167: This register contains the programmable scan limit when the special safeguard rung (TLET S8165) is not used. When the scan time in milliseconds exceeds the contents of this register, the processor halts and generate ERROR 970. Maximum allowable scan time is 32.765 seconds. If this register contains zero, the scan time is limited to one second. If this register contains a value larger than 1 second (1000), it automatically is reset to zero (i.e., 1 second) upon power up. Accuracy is within 2.5 milliseconds, rounded up.

8168: Communications monitor timeout. This register defines the time allocated for receiving a reply to a READ, WRITE, or ALARM rung. Bits 1-4 are used with the PRGMR (Channel 1) port, while bits 5-8 monitor the COMM (Channel 2) port.

The following table identifies the time interval range for bits in the 8168 register and is valid for both the PRGMR and COMM ports:

8168 Bits 1-4 and 5-8:

BIT PATTERN (4321 for PRGMR 8765 for COMM)	TIME COUNT IN MILLISECONDS
0000	0 to 250
0001	250 to 500
0010	500 to 750
0011	750 to 1000
0100	1000 to 1250
0101	1250 to 1500
0110	1500 to 1750
0111	1750 to 2000
1000	0 to 2000
1001	2000 to 4000
1010	4000 to 6000
1011	6000 to 8000
1100	8000 to 10000
1101	10000 to 12000
1110	12000 to 14000
1111	14000 to 16000

8169: This register assigns the BAUD rate for the Model 400's two communication ports. The first half of this register's data field is divided into two groups of 4 bits each. Each group of bits contains the baud rate for an individual port (channel).

The following table lists the channel-to-bit association for the PRGMR (Channel 1) port and the COMM (Channel 2) port.

8169 Bits 1-4 and 5-8

BIT PATTERN (1 to 4 for PRGMR, 5 to 8 for COMM)	BAUD RATE
0000	75
0001	110
0010	300
0011	1200
0100	2400
0101	4800
0110	9600 (default)
0111	19200

8170: Not used

8171: Contains the remainder of the most recent integer division performed. Remainder's sign (±) agrees with the numerator.

8172: Processor scan time in milliseconds. Scan time is the length of time required for the Model 400 to make one pass through user memory and includes end-of-scan I/O update and communication port servicing times.

8173: Fenced registers ending address (as entered by user). The *ending* point of the block of registers accessible via the COMM port (Channel 2).

8174: Fenced registers beginning address (as entered by the user). The *beginning* point of the block of registers accessible via the COMM port. Refer to Section 6.13 (security) for more details on the use of fencing.

8175: Error code storage. Refer to Appendix B for descriptions of these codes.

This register contains the last ERROR NUMBER which occurred. If there is more than one error, the most serious or highest priority error code is stored. The contents of register 8175 are cleared after a HALT-to-RUN keyswitch toggle, provided the cause of the error was eliminated. An exception to this are errors in serial communication, which clear as soon as a successful communication message has been exchanged through the indicated port. Errors occurring in register 8175 can be preserved in an "error register stack" using register 8107.

8176: Primary Processor Control Register. The following table explains what each bit in this register.

The individual bits of register 8176 are used for system monitoring and control. Because it is possible to alter the state of the data field bits, the user has the capability to change specified processor conditions. These bits can be altered via the CRT programmer or SFW Software's DATA ENTER mode or by a ladder rung in memory. Bits 1 through 16 are alterable. Exceptions to this involve the use of certain memory write protection modes. Refer to Section 6, "Software Security".

REGISTER 8176 READ/WRITE BITS - The following bits can be altered by the user:

Bit 1: When this bit is set ON, it prevents the Model 400 from scanning the ladder program. The processor can be placed in the HALT mode by either putting the processor keyswitch in the HALT position or by programming and energizing coil 8176-1 in the ladder diagram program. Toggling the keyswitch into and out of HALT or setting and resetting 8176-3 resets this bit to OFF.

Bit 2: When this bit is set ON, the processor operates in the DISABLE OUTPUTS mode. The processor can be placed in the DISABLE OUTPUTS mode either by putting the keyswitch in the DISABLE OUTPUTS position, or by programming and energizing coil 8176-2 in the ladder diagram program.

Bit 3: This bit operates as a remote HALT/RUN switch. 1 = Processor Halt, 0 = Processor Run. The HALT condition overrides the RUN condition. Putting the keyswitch in HALT or setting bit 3 to "1" halts the processor. To put the processor in RUN, however, the keyswitch must be in RUN or RUN/PROGRAM and bit 3 must be "0".

NOTE: ***Performing a DEL/CLR ALL operation does NOT reset bit 3; once set, it must be individually reset by the user.***

8176 Bits 1-32

BIT	USED FOR:
1	Processor HALT
2	Processor Disable Outputs
3	Processor HALT/RUN
4	Memory Protection Channel 2
5	Force Inhibit Channel 2
6	Register Protection Channel 2
7	Immediate I/O Update
8	IF Rung Overflow Flag
9	Control Register Data Corruption Flag
10	MATRIX or ARRAY Overflow Flag
11	Immediate Communications Update, Channel 1
12	Immediate Communications Update, Channel 2
13-14	**Not used**
15	Port and Route Edit Lockout Control
16	Inhibit Coil Address
17	Either Internal or Power Supply Battery is Low
18	OPEN Contact Address
19	SHORT Contact Address
20	*Not used*
21	First Dummy Scan
22	Second Dummy Scan
23	Normal Program Scan
24-31	**Not used**
32	Internal Battery Low

Bit 4: Setting this bit to "1" protects user memory from being altered via the COMM port (Channel 2). For further details refer to Section 6.10.
1 = Memory Protected
0 = Memory Accessible.

Bit 5: Setting this bit to "1" inhibits the forcing of any I/O via the COMM port. 1 = Force Inhibit, 0 = Forcing Allowed. I/O can still be forced using programming equipment through the PRGMR port (channel 1). Refer to Section 6.11.

Bit 6: In addition to the fencing capabilities of the Model 400, setting this bit protects any registers from being altered via the COMM port. Registers can still be forced or altered through the PRGMR port using programming equipment. 1 = register protect, 0 = register accessible. Refer to Section 6.12.

Bit 7: Activates the Immediate I/O Update feature of the Model 400. When the Model 400 encounters a rung containing an energized bit 8176-7, it momentarily stops scanning the ladder program and updates the external I/O that are defined by registers 8105 and 8106 (inputs are written to the image table, outputs are read from the image table). Upon completion of I/O updating, normal scanning resumes. If either 8105 or 8106 is set to "0", only the local digital I/O are updated (this default does not apply to local register modules).

Bit 8: The Model 400 is capable of complex COMPARE functions (i.e. IF S50 = S60 X S70...). If the integer or floating point math operation within a COMPARE statement results in a value beyond the processor's capability, bit 8 is set to "1". This bit is reset to "0" at the beginning of each new scan.

Bit 9: This bit is set if the Model 400 detects a control register checksum error upon power-up. ERROR 900 or 901 appears in register 8175 unless overwritten by a higher priority error. Also indicates that all control registers have been set to their default values and that all data storage registers have been reset to "0". Typically this bit is set if battery backup has failed for any length of time. Bit 9 is reset by a CLEAR ALL or DATA ENTER operation.

Bit 10: This bit is set to "1" if an overflow occurs during a MATRIX or ARRAY operation and is reset at the beginning of a new MATRIX or ARRAY rung.

Bit 11: When this bit is set to "1", the Model 400 interrupts the normal ladder scan and performs up to 5 milliseconds of servicing on any pending communication in the buffer for the PRGMR (Channel 1) port. Normal ladder scan continues upon completion of the communication servicing.

Bit 12: Performs the same function as bit 11, except for the COMM (Channel 2) port.

Bit 13/14: Not used.

Bit 15: Setting this bit to "1" prevents program editing through either port and any route by any device other than the device which set the bit to "1". Refer to Section 6.5.

Bit 16: This "inhibit coil rung" can be placed within the main ladder program to prevent alteration and/or viewing of certain rungs. Refer to Section 6.7 for more details.

REGISTER 8176 READ-ONLY BITS - The following bits cannot be altered by the user in any way, however, these can be examined as contacts in the program:

Bit 17: This bit is set to "1" when either the Model 400's internal lithium battery or the power supply batteries are low. This bit can be used in conjunction with bit 32 of this register which signifies that only the internal lithium battery is low.

Bit 18: "OPEN". This bit is always de-energized, and is used to signify an OPEN in the ladder program.

Bit 19: "SHORT". This bit is always de-energized. A normally-closed contact in the ladder program uses this bit to signify a SHORT.

Bit 20: Reserved for processor internal use.

Bit 21: Set to "1" during the first power-up dummy scan. In a processor that is revision 1, this bit is not used.

Bit 22: Set to "1" during the processor's second power-up dummy scan.

Bit 23: This bit is set to "1" whenever the Model 400 is scanning the ladder program after executing the first two dummy scans.

Bits 24, 27 to 31: Not used.

Bits 25, 26: Reserved for internal use.

Bit 32: Set to "1" when the voltage of the Model 400's internal lithium battery falls below the point needed to maintain memory in RAM. This bit can be used in conjunction with bit 17 to provide a better idea of which battery is defective or missing.

8177: Password register. Contains a value that determines if register 8178 controls editing, forcing, and register write restrictions. If a zero is in register 8177, then any value in register 8178 is ignored. When reading register 8177, if a zero is stored then a zero is displayed. If any other value than zero is stored in register 8177, then "-1" is displayed to keep the password a secret. See Section 6 for more details.

8178: Restriction register. Each of the first 8 bits in this register determines which security feature is enabled for the PRGMR or COMM port. See the following table to determine which security is in effect when the indicated bits are set to "1".

8178 Bits 1-8:

BIT	SECURITY MEASURE IN EFFECT WHEN BIT IS SET TO "1"
1	Restricts priority writing to registers through the PRGMR port.
2	Restricts the forcing of registers through the PRGMR port.
3	Restricts *non*-priority writing to registers through the PRGMR port.
4	Restricts program editing through the PRGMR port.
5	Restricts priority writing to registers through the COMM port.
6	Restricts the forcing of registers through the COMM port.
7	Restricts *non*-priority writing to registers through the COMM port.
8	Restricts program editing through the COMM port.

NOTE: ***The CLEAR ALL operation is always allowed through the PRGMR port.***

8179: Not used.

8180: Percentage of memory used on high-speed scanning processor (compiled user memory).

8181: Indicates the amount of ladder program memory (MSW) in bytes that is available for use. This figure reflects intermediate memory.

8182: Indicates the amount of ladder memory (LSW) in bytes that is available for use. This figure reflects intermediate memory.

8183: Stores information on some types of errors. See Appendix B.

8184: Same as 8183. See Appendix B.

8185: Indicates the number of rack addressing rungs that are programmed into the Model 400.

8186: *Primary Processor Status Register.* The individual bits of this register are used for indicating certain processor conditions. These bits can be examined, but cannot be altered in any way. See the following table for an explanation of each of the bits in register 8186.

8187: Indicates the number of ladder rungs that are programmed in user memory.

8188: The three most significant digits in this register identify the *type* of Model 400 (i.e., 401, 423, 424, or 444). The least significant digit identifies the major revision level. In Revision 4 and later, the 16-bit status field stores both the major and minor firmware revision levels in Binary Coded Decimal (BCD) format. The two most significant BCD digits represent the major revision level, while the two least significant represent the minor revision level.

8189: Memory word size of the processor in bytes. Bits 1 through 8 indicate memory available (MSW). Bits 9 through 16 indicate the number of bytes that make up a user word.

8190: Total intermediate memory size in bytes; *Least* Significant Word (LSW).

8191: *Intermediate memory used in bytes; Most* Significant Word (MSW).

8192: Intermediate memory used in bytes; *Least* Significant Word (LSW).

8186 Bits 1-16:

BIT	MODEL 400 STATUS WHEN BIT IS SET TO "1"
1	Halted
2	Outputs disabled
3	*Running*
4	MEMORY LED is ON
5	FORCE LED is ON
6	I/O LED is ON
7	Key is in HALT position
8	Key is in DISABLE OUTPUTS position
9	Key is in RUN position (if bits 7, 8, and 9 are "0", then key is in RUN/PROGRAM position)
10	WRITE PROTECT LED is ON or FLASHING (hardware or software security is enabled)
11	BATTERY LOW LED is ON or FLASHING.
12-16	NOT USED

14 TECHNICAL DATA

This section provides supplementary technical details on memory use, scan speed, and general processor operation.

14.1 Memory Utilization

The Model 400 contains *two* copies of the user memory: an *intermediate code* memory and a *compiled code* memory. Intermediate code is the "universal language" that all SY/MAX programmers and processors "speak", while compiled code allows for fast program execution.

Intermediate code memory is the limiting factor for user program size. The "Memory Available" and "Memory Used" displays on the status screen of the programming device reflect intermediate memory. In this section, the terms "user memory" and "memory" refer to intermediate memory. Refer to Section 14.3, "Theory Of Operation", for additional information.

The following sections explain memory use for Relay Circuits, General Instructions, and IF/LET/MATH Instructions.

14.1.1 RELAY CIRCUITS

In general, the amount of memory required for relay circuits can be determined by counting the number of contacts, parallel branches, and coils in each rung. A contact or coil, including its address number, requires one word of memory. Spaces do not require *any* words of memory.

The sample rung in Figure 14.1 requires eleven words of memory. The numbers indicate the associated words for each contact and output:

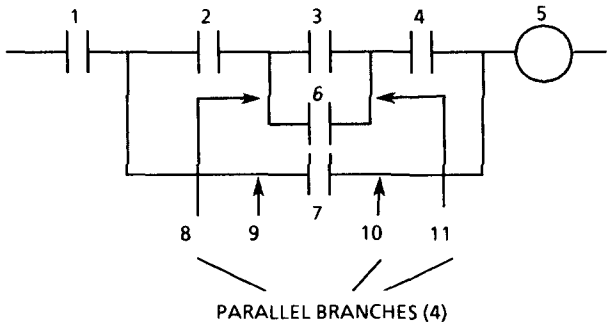


Figure 14.1 Rung Using 11 Words of Memory

Each rung of memory also requires an additional 1/3rd of a word. *This 1/3rd word cannot be represented by the "Memory Used" display on the programmer.*

For example, if a contact (one word) and a coil (one word) are added to user memory, the programmer may indicate that *three* words of memory were consumed. The third word here is the fractional (1/3rd) word rounded up to a whole word.

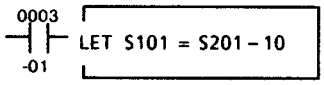
14.1.2 GENERAL INSTRUCTIONS

Figure 14.2 shows memory use for most Model 400 instructions. See Section 14.1.3 for special considerations on the IF, LET, and math functions.

14.1.3 IF, LET, AND MATH INSTRUCTIONS

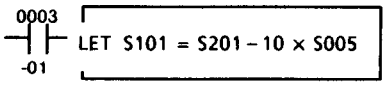
In an IF or LET instruction, the amount of memory required is a function of the type(s) of operation being performed. From the table in Figure 14.2, any IF or LET statement (including matrix) requires a fixed quantity of words for the basic instruction, plus one or more additional words for each addition, subtraction, AND, OR, or EXCLUSIVE OR performed in the horizontal box.

The following sample rung illustrates this memory use:



This rung requires a total of five words of memory: one word for the contact, three words for the basic LET itself, and one word for the subtraction operation. *If the above rung were changed to a MATRIX LET, the memory consumed would increase to seven words (as shown in Figure 14.2, a matrix transfer requires five words of memory).*

Each additional math operation added to the LET box increases the memory consumption as shown in the following rung:



Since this LET box now contains a multiplication, the rung requires a total of six words: three for the basic LET, one for the contact, and one each for the subtraction and multiplication operations.

INSTRUCTION	MEMORY USED
GENERAL RUNGS	
Contact	1
Parallel Branch	1
Coil	1
Coil, Latch/Unlatch	1
Coil, Transition	1
Master Control Relay	1
Timer	6
Counter	6
Shift Register (sync. or asynch)	6
SUBROUTINES	
MARK/GOTO/Start of Subroutine	4
GOSUB	4
RETURN	3
PRIORITY COMMUNICATIONS	
Register READ/WRITE	5 ▸
ALARM Message	4 ▸
Print ASCII	2 ▸▸
Immediate I/O or Communication Update	1
LET, IF, AND, OR	
LET (data transfer)	3
LET (matrix data transfer)	5
IF (compare)	3
IF (matrix compare)	5
AND / OR / EXCLUSIVE OR	1
Binary-to-BCD, BCD-TO-Binary	1
INTEGER MATH	
Addition /Subtraction	1
Multiplication /Division / Square Root	1
FLOATING POINT MATH	
Addition/Subtraction/Multiplication/Division/Square Root/Sine/Cosine/Base 10 and Base E Logarithms/Y to the X	1*

* If any number in a LET or IF box is a floating point constant, then two words of memory are required.
▸ Add one word for the baud rate and one word per route.
▸▸ Add one workd per register or one word for each two characters.

Figure 14.2 Memory Use

14.2 Scanning Techniques

14.2.1 DESCRIPTION

The Model 400 Type SCP-423, 424, and 444 contains a high-speed Scan Processor, Math Coprocessor, and Control Processor. The Scan Processor executes certain types of tasks in parallel and operates faster than the Control Processor for better throughput. The Control Processor in the SCP-401 performs both math and scanning operations.

The Model 400 Processor scans the ladder program in two separate phases:

1. Scanning of the ladder program
2. I/O Update (Including forcing, if active)

The Model 400 makes use of an Image Memory to hold the state of all I/O and registers. The contents of the Image Memory are transferred to the external I/O. The present I/O state is transferred to the Image Memory at the end of every scan.

The time necessary to perform an I/O update is a function of the number of I/O registers assigned to other modules in the system and how many of these registers are fragmented (containing both input and output data in the same register).

14.2.2 SCANNING SPEED

In addition to the time needed to execute the ladder program, the *scan time* of the Model 400 is composed of the time needed to update the Image Memory *plus* up to five milliseconds per scan to service the communication ports. In general, the following equation can be used to estimate an approximate scan speed:

$$ST = [K \times (LUP)] + [0.04 \times (ER)] + [0.45 \times (FER)]$$

In the above equation:

ST is the SCAN TIME in milliseconds.

K is the LADDER SCAN SPEED per K of user program (estimated from execution times in Figure 14.3 on the next page).

LUP is the LENGTH of USER PROGRAM in thousands of words (k words).

ER is the number of EXTERNAL REGISTERS.

FER is the number of FRAGMENTED EXTERNAL REGISTERS

Figure 14.3 provides a list of typical operations and an approximation of associated execution times. For operations not listed in Figure 14.3, execution times are difficult to estimate because of the complexity of certain operations. For example, matrix IF and LET operations are of variable length and functionality, which can impact scanning time significantly.

WARNING

When editing while the processor is executing the program, the program scan time may increase significantly as discussed below.

When performing an edit while the processor is executing the user program, the normal processor scan time may be momentarily increased. The increased scan time is a one-time occurrence and is caused by the processor loading the new rung(s) into the location in memory where the program resides.

Depending upon the program complexity, the type of edit performed, and the memory location where the edit takes place, the scan-time may increase several hundred milliseconds. The longer and more complex the program (i.e., multiple MARK, GOTO, and SUBROUTINE commands) and the earlier in memory that the edit takes place, the greater the scan-time increase.

For any given program, the worst-case scan time can be measured by monitoring register 8172 (scan-time register) while performing an insert of rung 1. If an excessive increase in scan time cannot be tolerated, register 8167 (programmable scan limit) may be preset to halt program execution based on a user-specified maximum scan time.

INSTRUCTION TO BE EXECUTED	EXECUTION TIME IN MICROSECONDS (Except where noted)	
	Types SCP-423, 424, 444	Type SCP-401
Contact*	0.7	3.7
Coil (Including MCR and latch/unlatch)	1.3	8.5
Transitional Coil	1.3	11.5
Timer (Disabled/Enabled)	14.0/17.7	39.5/45.0
Counter (Disabled/Enabled)	19.0/21.3	54.2/65.2
IF* (Integer/ Floating-point)	11.5/110	21.5/n.a.
LET* (Integer/ Floating-point)	9.5/108	26.2/n.a.
Matrix LET* (Integer)	89 per register	n.a.
Math (Integer/Floating-point):		
ADDITION	5.0/43.0	5.8/n.a.
SUBTRACTION	5.0/43.0	5.8/n.a.
MULTIPLICATION	119/39.0	42.3/n.a.
DIVISION	119/40.5	41.3/n.a.
Logical AND, OR, XOR	3.5	1.7
Shift	83.0 + 14 per register	51.5 + 11 per register
FIFO	87.5 + 14 per register	56.5 + 11 per register
PID* (Rev. or Dir.)	675 (typical)	n.a.
PID* (Manual)	113 (typical)	n.a.
Immediate I/O Update†	130 + 40 per register	89 + 40 per register
T WRITE, T READ, T ALARM	131	111
T PRINT	181 (typical)	134 (typical)
SIN/COS/TAN	234	n.a.
Ln/LOG	144	n.a.
Y ** X	346	n.a.
Add Stat	328	n.a.
Variance	268	n.a.
MARK ST SUB	0	0
MARK	0.5	2.7
GOTO*	1.3	6.0
GOSUB* (with RTN)	69.0	51.1

Figure 14.3 Execution Times For Typical Operations

INSTRUCTION TO BE EXECUTED	EXECUTION TIME IN MICROSECONDS (Except where noted)	
	Types SCP-423, 424, 444	Type SCP-401
External I/O Update Time †	320 per 8 local registers per scan	320 per 8 local registers per scan
MIX #1**	0.7 milliseconds per K of ladder	4.1 milliseconds per K of ladder
MIX #2***	2.9 milliseconds per K of ladder	7.9 milliseconds per K of ladder
MIX #3****	2.5 milliseconds per K of ladder	7.0 milliseconds per K of ladder

Figure 14.3 Execution Times (continued)

FIGURE 14.3 CODES

† *123 microseconds per register for Class 8030 Type RIM121 analog input, 75 microseconds per register for Class 8030 ROM121 analog output.*

* *If rung is TRUE up to that point; if FALSE, the instruction is not executed and time is reduced. See Section 14.2.3, "Rung Execution Algorithm Guidelines".*

** *Mix #1 is composed of all Boolean rungs.*

*** *Mix #2 is 50% Boolean, 34% IF and LET rungs, 8% counters, and 8% timers.*

**** *Mix #3 is 80% Boolean, 5% timers, 5% counters, and 10% addition/subtraction.*

CONSIDERATIONS FOR "TIME MIXES" #1-#3

- On an average, only 66% of any rung must be executed.
- Boolean assumes an average of two contacts for each coil.
- External I/O update time assumes eight unfragmented external registers.
- External I/O Update time is not included in any of the mix execution times. At the end of each scan, up to five milliseconds can be used for Model 400 communication processing tasks.
- Times given for the SCP-401 assume that the processor is handling communication interrupts about 30% of the time.

14.2.3 OPTIMIZING SCAN SPEED

In some ways, optimizing the scan speed of a Model 400 processor is no different than the procedure used for any computing device. For example, the Model 400 program is compiled from intermediate code into a 64-bit word compiled format that allows rapid execution by the scanning processor.

While this intermediate to 64 bit compiled code conversion operation is transparent to the user, the Model 400's instruction set contains instructions like GOTO and GOSUB that allow skipping over sections of a program that does not require scanning at a particular time. This "skipping over" process saves scan time and improves performance (throughput). This simple technique should not be overlooked when dealing with programs containing sections that do not require constant scanning.

When laying out and programming any system, applying knowledge of how that system behaves to the system's configuration can optimize its performance. On the next page are some simple guidelines (most of which have been previously discussed in this manual) for economizing Model 400 scan time.

RACK ADDRESSING GUIDELINES

Optimizing scan speed using good rack addressing techniques is demonstrated by the example in Section 3.6.5. When Rack Addressing a system containing Local Interface (LI) module(s), scan speed is optimized by assigning *only those registers necessary for remote I/O operation* to an LI. Any registers (up to 255 - recall that registers above 255 aren't transferred) unnecessarily assigned to an individual LI results in needless end-of-scan register transfers between the Model 400 and the LI's data storage registers.

The equation of Section 14.2.2 shows that updating an unfragmented register on the bus requires 40 microseconds. If the conditions of the example in Section 3.6.5 are used, the unnecessary assignment of all 512 LI module registers decreases throughput as shown below:

Update Time for 512 registers =
255 x 0.04 ms = **10.2 ms per scan**

Update Time for 50 registers =
50 x 0.04 ms = **2.0 ms per scan**

SUMMARY: In the scan time calculations, 8.2 milliseconds is needlessly added to the scan time. Thus, assign only needed registers for external I/O to the LI module(s).

When assigning registers to other register modules in the CPU rack, ONLY assign as many registers as needed by the module. For example, assign only one register to a 16-point digital I/O module, or four registers to a four-channel analog module. Any registers needlessly assigned to the module causes the 40 microsecond per register scan penalty.

EXTERNAL I/O REGISTER GUIDELINES

The equation of Section 14.2.2 shows that the time required to update a register increases tenfold when the register is *fragmented* instead of unfragmented. A fragmented register refers to an external I/O register that contain both input and output points. Thus, when using four-point and eight-point digital I/O modules, lay out the programmable control system so that *all* points of a particular external I/O register are assigned as *either* inputs *or* outputs.

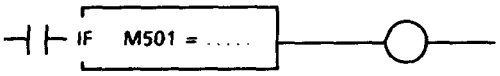
For example, if a Model 400 is being used in an HRK-type rack (eight-function rack), both modules associated with an external I/O register should be either inputs or outputs. This principle applies to remote I/O registers as well.

NOTE: Fragmenting registers forces the processor to make decisions that consume valuable processing time. These time-consuming decisions can be eliminated if external I/O registers are made up strictly of inputs or outputs.

RUNG EXECUTING ALGORITHM GUIDELINES

The Model 400 executes ladder rungs in such a way that elements of conditional logic may actually be skipped over if the logical outcome of the rung is no longer in doubt. This applies to logically TRUE as well as logically FALSE rungs.

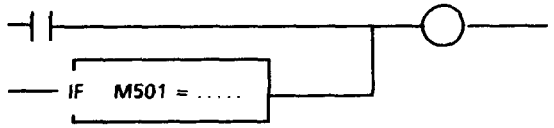
A good example of this is shown in the rung below:



The matrix IF instruction shown above is only be executes if the leading contact is CLOSED. If OPEN, the matrix IF is skipped over (no pointer information is updated) and the coil is turned OFF. If the leading contact is CLOSED, the matrix IF is scanned to determine the state of the coil.

If the order of the conditional elements was inverted (i.e., the matrix IF came *before* the contact), although logically equivalent, the execution time could increase dramatically for the FALSE condition since in this configuration the matrix IF would *always* be executed. If proven FALSE, the contact is skipped over; the savings in scan time, however, would be negligible as compared to the savings if the matrix IF were not scanned at all.

A similar principle applies to saving processor time for the TRUE condition shown below:



If the contact shown above is closed, the coil is ON regardless of the condition of the matrix IF instruction. In this case, the matrix IF doesn't need to be executed and some scan time could be saved in the process.

However, if the matrix IF and the contact were interchanged, the matrix IF always executes first. This may have an adverse effect on the rung's execution time.

Another simple rung execution technique can be applied to LET statements. Since the LET only executes if the rung is TRUE, any initialization constants scanned more than once will needlessly consume processing time and should be avoided in the user program.

SUMMARY: When placing a premium on optimizing scan time, examine rungs for logical equivalence to see if interchanging conditional elements will reduce the execution time.

SCANNING PROCESSOR VERSUS CONTROL PROCESSOR GUIDELINES

Any functions that *cannot* be handled by the scanning processor must be "handed off" to the control processor. Since the control processor is also tasked with handling other system interrupts, longer scan times can result.

Such multitasking operation makes it very difficult to quantify precise execution times for these instructions since it cannot be predicted in advance how many other demands (such as communication handling) are being made on the control processor at the time of the "hand-off".

As shown in the table of Figure 14.3, execution times can vary widely. This variation occurs because some of the instructions are executed by the scan processor while others must be handled by the control processor. For example, integer addition and subtraction are handled by the scan processor, while integer multiplication division and all floating-point operations are handled by the math coprocessor.

Since integer multiplication and division and floating-point math operations are handled by the *math* coprocessor, the *control* processor must act as an intermediary between the scanning processor and the math coprocessor. While this is occurring, *the scanning processor remains idle until the result* is computed. This make for an inefficient use of processing power. Applying the principles of the "RUNG EXECUTING ALGORITHM" section, time is saved when executing floating-point IF and LET statements.

The following provide additional considerations when comparing the operation of the scanning processor with the control processor:

- Although GOSUB and RETURN instructions have to be executed by the *control* processor, their careful use can result in a great savings in throughput.
- Shift register and FIFO operations are executed by the *control* processor; their execution times are affected by the number of registers involved.
- *Matrix operations must be performed by the control* processor and can significantly influence scan time depending on their size and pointer configuration:
  - When using *matrix LET* statements, use "level" or "incremental" LETs over "repeat" LETs wherever possible. This distributes the increased processing time over many scans.
  - For *matrix IFs*, use "level" instead of "repeat" whenever possible. If "repeat" *must* be used, follow the principles given in the *Rung Execution Guidelines* section to save scan time.

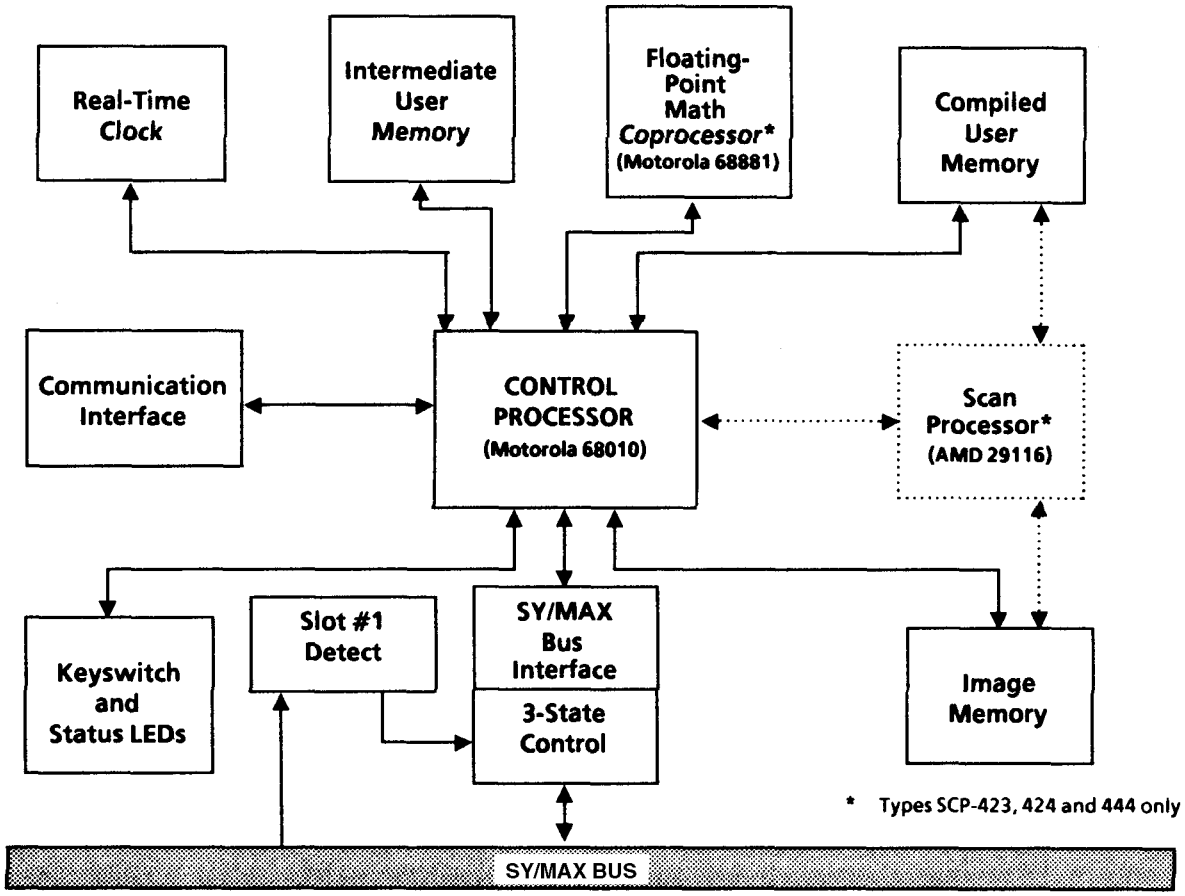


Figure 14.4 Model 400 Block Diagram

14.3 Theory Of Operation

The Model 400 consists of several major subsystems, all under the control of a central processing unit. Refer to the block diagram in Figure 14.4 while reading this section.

The control processor is a Motorola Type 68010 and is responsible for performing and/or coordinating all functions of the Model 400, including interrupts and error conditions from the communications interface, SY/MAX bus, scan processor, and math co-processor.

Scanning in all Model 400 Processors, except the Type SCP-401, is performed by an Advanced Micro Devices Type 29116 processor utilizing a 64-bit compiled word.

Floating point operations are performed by a Motorola Type 68881 math coprocessor.

Model 400 Types SCP-423, 424, AND 444's system clock runs at 8 MHz clock. The SCP-401's system clock runs at 10 MHz clock.

Each subsystem within the Model 400 is briefly explained in the following.

Control Processor: This device either performs or coordinates all of the processor's functions, including communication via the two serial RS-422 ports. The *control processor also compiles the ladder program* in the Compiled User Memory, controls interrupts and error conditions, and handles SY/MAX bus duties.

NOTE: *In the SCP-401, the control processor also serves as the scan processor.*

Math Coprocessor (Processor Types 423, 424, and 444 only): Allows the Model 400 processor to *perform floating-point math operations.*

Scan Processor (SCP-423, 424, and 444 only): Performs computation of the Model 400's output states based on the current condition of inputs and registers. The computations and sequence in which they are performed are determined by the Compiled User Memory. The I/O states and register values are obtained from the Image Memory.

Image Memory: The Image Memory consists of battery-backed RAM that provides 16-bit data and status fields for user registers 1 through 4000 and control registers 8097 through 8192. Parity protection is provided for this memory. This memory contains both the internal and external I/O *and registers.*

Between scans, the control processor outputs the current values in this memory to the appropriate external devices. The control processor also updates the Image Memory according to the external I/O/registers to prepare for the next scan.

User Memory: Partitioned into two distinct portions for maximum efficiency in scanning and program manipulation:

1. *Intermediate Code User Memory*: This battery-backed memory occupies a maximum of 104k bytes of the control processor's operating (scratch) memory. The Intermediate User Memory is only accessible by the control processor. It serves as the source for the compile operations which generate the instructions for the Compiled User Memory.

2. *Compiled Code User Memory:* Consists of battery-backed CMOS static RAM that provides 64-bit word user program storage. Each word is composed of the scan processor instructions and control codes. This memory serves as the executive memory for the scan processor and can be randomly accessed by the control processor for program loading and editing purposes.

Communication Interface: Controls the two RS-422 ports on the front of the Model 400 (PRGMR/Channel 1 and COMM/Channel 2). More information on these ports is found in Section 5.

SY/MAX Bus Interface: Allows the Model 400 to control a SY/MAX digital or register rack as a bus master.

Real-Time Clock/Calendar: Provides the Model 400 with time-of-day and date information. This information can be used throughout the entire programmable controller system. For more information on the clock see Section 12.2.

14.4 Forcing

WARNING

Since forcing overrides the actual logic of the control program, ONLY QUALIFIED PROGRAMMERS who understand the contents of this section should use the forcing capability.

Forcing allows an external input or output to be turned ON or OFF via the programmer's keyboard and overrides the ladder logic state of the external field device.

In the Model 400, forcing is implemented just prior to updating the external I/O at the end of scan. *The forced state of an output WILL NOT be reflected in the image table, meaning that the user program logic executes and is based on the actual image table logic state and not on the forced state.* This operation differs in comparison to other SY/MAX processors for the following:

The forced state of an input is written to the image table. Since forcing is only implemented in the external I/O at the end of scan, forcing an internal relay or data storage register does not alter the actual logic state in the user ladder program. The programmer indicates the bits of an internal relay or data register to be forced even though they are *not* forced. This operation differs from other SY/MAX processors and is discussed in the following.

Since forcing is implemented on external I/O and *not* in the image table, the following rules govern *what is observed* and *how the programmable controller system behaves.*

FORCING AN EXTERNAL OUTPUT

A forced external output causes the associated external field device to assume the forced state. The programming device reflects this when displaying a forced rung by displaying an "F" within the forced element. If relay contacts with the output address are used in the program, they will assume the logic state of the output as executed in the image table. The relay contacts and coil displayed on the programmer reflect the forced state and *not* the image table logic state, this may conflict with the way the logic is actually being executed.

EXAMPLE: Consider the rungs shown in Figure 14.5. Contact 13-02 is an external input, while coils 14-16, 13-11, and 13-12 are external outputs. The following conditions apply to the rungs:

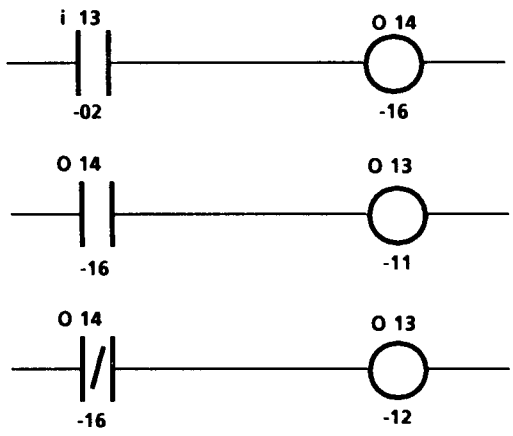


Figure 14.5 Forcing Example

CONDITION: Output **14-16** is forced ON, while input **13-02** is OPEN.

RESULT: The external device wired to output **14-16** is energized, and the programmer needs to insert an "F" inside the coil and relay contacts with address **14-16.** The programming device displays the relay contacts and coil that is ON to reflect the forced state.

COMMENTS: Even though the N.O. relay contact **14-16** is forced "ON", output **13-11** is OFF while output **13-12** is ON. This occurs because coil **14-16** and its associated relay contacts are actually OFF in the image table (because contact **13-02** is OPEN).

If contact **13-02** is CLOSED instead of OPEN, the image table logic agrees with the forced state; output ***13-11** is ON while output **13-12** is OFF.*

If it is desired to have the relay contacts of coil **14-16** follow the logic state of the coil, the proper technique is to force external input **13-02** as discussed in the following:

> **WARNING**
>
> **Forcing an unlatch coil "ON" actually latches the coil.**

FORCING AN EXTERNAL INPUT

A forced external input overrides the actual condition of the input field device and causes the associated bit address in the Model 400's image table to assume the forced state. The programming device indicates the forced state. As long as the forced external input bit is not altered in the ladder program (can only happen if the input bit was incorrectly programmed as an *output*), the bit state is used when the ladder program is executed.

EXAMPLE: Assume the same rungs are used in this example as in the previous "EXTERNAL OUTPUT" example.

CONDITION: Input **13-02** forced ON, external field input is OFF.

RESULT: As long as the state of bit **13-02** is not altered in the user ladder program, the forced state is preserved in the image table no matter what the state of the external field device's input is. This means that output **14-16** is actually turned ON *in the image table* along with the external device connected to output **14-16.** The relay contacts of coil **14-16** properly reflect the state of the coil. Output **13-11** is ON, while output **13-12** is OFF.

FORCING AN INTERNAL RELAY EQUIVALENT OR DATA STORAGE REGISTER

As discussed previously, forcing is a condition which applies only to *external* I/O.

NOTE: ***Attempting to FORCE an internal relay equivalent or a bit in a data storage register has no effect on its state in the image table when used to execute the ladder program. This is true even though the programming device displays the FORCED state.***

Since there is no external I/O associated with a forced internal relay equivalent or storage register bit, the Model 400 has no way of implementing the forced state. However, the programming device shows the bit as forced; *this may conflict with the way the rung is actually being executed.*

This situation is somewhat analogous to the behavior of contacts that use the same address as a forced coil. The contacts *indicate* a forced state, but in actuality assume the logic state of the output as executed in the image table. In the case of an internal relay there is no external output associated with the address. The Model 400 has nothing to force externally and the internal relay contacts *continue to follow the logic of the coil as instructed* by the user program.

To summarize, perform forcing *ONLY on external inputs and outputs.* Forcing an *internal* relay equivalent or bit in a data storage register has no effect on the register and can produce information that is subject to misinterpretation on the programming device screen.

APPLICATION CONSIDERATIONS FOR FORCING

- Any external I/O register is forcible. This includes local digital I/O in a register rack and digital I/O controlled via the Parallel Digital Driver/Receiver (PDD/PDR) modules.
- Bits in a maximum of 256 registers can be forced simultaneously.
- Forcing can *always* be cleared by cycling power to the Model 400, by powering down all SY/MAX devices connected to both serial ports, or by disconnecting the devices from both serial ports.
- Forcing can be inhibited or if already in effect, prevented from being altered by placing the keyswitch in the RUN (as opposed to RUN/PROGRAM) position. In this mode, forcing *is* cleared if both serial ports are disconnected or the attached SY/MAX devices are powered down.
- When the TIMED INTERRUPT subroutine is programmed and the Model 400 is in the RUN/PROGRAM mode, forced I/O *can* be released by the *FORCE - CLEAR ALL* command, by disconnecting the serial ports or by removing power to the devices connected to the serial ports.
- Software security can be invoked to restrict forcing on a port basis; refer to Sections 6.9 and 6.11.
- Each forced register adds approximately 100 microseconds to the I/O update time at the end of the processor's scan, regardless of the number of bits forced in the register. Thus, forcing *one* bit in a register causes the same time penalty as forcing all 16 bits in that register.

14.5 Power Up/Down Sequence

PROCESSOR POWER-UP SEQUENCE

Upon application of power to the Model 400, the following start-up sequence occurs:

1. External outputs are turned OFF.
2. A self test sequence is performed to determine proper operation. If any part of the self-test sequence fails, the Model 400 will not go into RUN.

The following components are checked when the self-tests are run:

- The executive PROM; checksummed and validated
- User memory
- Intermediate User Memory
- Internal RAM
- Compiled User Memory
- Image Memory
- Scan Processor operation
- Control Processor operation
- Watchdog Timer
- Math Coprocessor (if equipped)

3. Memory Corruption Test. If an error is found, the MEMORY LED is turned ON and the Model 400 does not execute the existing programmed memory.
4. A minimum 1-second delay is generated in order to allow I/O data to stabilize. The delay may be longer depending on the register cards installed in the processor system and the quantity of memory in the processor.
5. The I/O forcing memory is reset inside the Model 400, thereby releasing all forced I/O.
6. Input status is updated in the Image Memory.

NOTE: *The total time required for all power-up sequence activities is about seven seconds. During this time, the Model 400 does not respond to any communication activity, and any attached programming device appears to be inactive (in reality, the programming device is being requested to "wait" by the Model 400).*

7. If the processor keyswitch is in the RUN, RUN/PROGRAM, or DISABLE OUTPUTS position, the sequence continues with the "Processor Scan Start-up Sequence" described in the following.

PROCESSOR SCAN START-UP SEQUENCE

Upon changing the processor keyswitch from any position to either the DISABLE OUTPUTS, RUN, or RUN/PROGRAM position, the following sequence occurs:

1. A self-test is run which checks that:
   - A valid user program is loaded
   - All user-programmed MARKs and related GOTO and GOSUB rungs are properly programmed.
   - All programmed I/O devices are checked
2. Two dummy scans are conducted:
   - *Dummy Scan #1* simulates the presence of an MCR in the OFF state and executes the scan so that DISABLE OUTPUTS is ON. Subroutines are scanned first, then the regular ladder program is scanned. This scan has the effect of turning OFF all coils, except for those programmed as internal latches. Outputs in the image table are not transferred to the I/O devices during this scan. The timed interrupt is not executed.
   - *Dummy Scan #2* removes the simulated MCR, but keeps DISABLE OUTPUTS set ON. The ladder is scanned normally. During this scan, timed interrupts are not executed. At the end of this scan the image table is transferred to the external I/O and programmable interrupt scheduling begins. During this scan, bit 22 of register 8176 is set to "1".
3. Normal ladder program scanning begins. Bit 23 of register 8176 is set to "1".

NOTE: *The total time for keyswitched start-up activities is about three seconds. During this time the Model 400 does not respond to any communication activity. An attached programming device appears to be inactive (although in reality its simply being asked to "wait" by the Model 400). If the programmable control system includes a serial interface module (LI, LTI, or LAI), start-up time may be up to 15 seconds if the serial interface is showing an error condition.*

PROCESSOR POWER-DOWN SEQUENCE

When the Model 400 processor enters a halted state or if power is removed from the system, the Model 400 performs an orderly shutdown procedure that resets all external outputs to their OFF state and halts the memory scan. If Local Interface (LI) modules are used in the system, outputs can be specified to *FREEZE* instead of turning *OFF*; refer to the Local Interface Instruction Bulletin #30598-247-XX.

The programmable controller system power supply continues operating, with no interruptions in the operation, through any power loss or "brown-out" condition of approximately 16 milliseconds or less. After this period of time, the processor shuts down and restarts when proper power is resumed.

In addition to all external outputs being reset to OFF in the modules, the *registers* that are rack addressed to the output modules are also reset to 0 in the processor image table. This occurs not only during the CPU power-down sequence, but also during run-to-halt transitions. However, in Revisions 1 and 2 of the Model 400, registers which are rack addressed to 16/32 function digital output modules are retentive. If programmed as coils, such behavior causes no operational difference unless programmed as LATCH/UNLATCH coils.

Figure 14.6 provides a rung example where the outputs behave the same when scanning resumes regardless of the output module type.

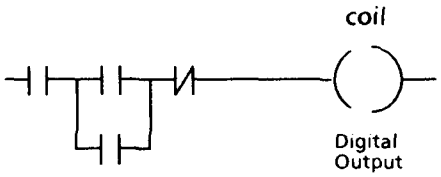
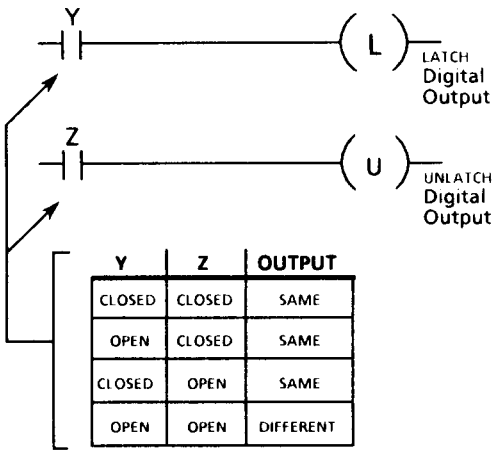


Figure 14.6 4/8 and 16/32 Function Digital Outputs Behave the SAME

In Figure 14.7, the 4/8 and 16/32 function outputs behave the same unless both the LATCH and UNLATCH rungs are false (both contacts Y and Z OPEN). In this case, 4/8 function LATCH outputs are reset to OFF, while 16/32 function LATCH outputs retain the state that existed at the time the processor halted.

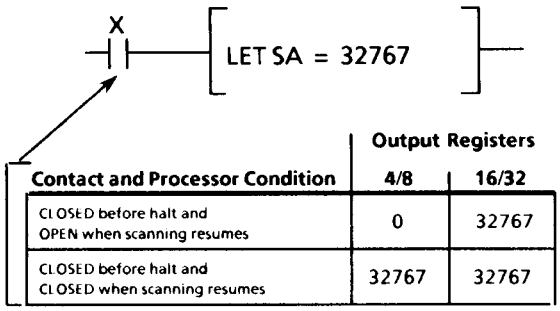


Y	Z	OUTPUT
CLOSED	CLOSED	SAME
OPEN	CLOSED	SAME
CLOSED	OPEN	SAME
OPEN	OPEN	DIFFERENT

Figure 14.7 4/8 and 16/32 Function LATCH and UNLATCH Digital Outputs Behave the SAME

If outputs are driven directly (i.e., have their ON or OFF state defined) by bits of a register, 4/8 function digital outputs reset to OFF and remain OFF until the register that defined the state is updated by the user program. In Revisions 1 and 2, 16/32 function digital outputs are **not** reset, but retain the state defined by the register value in the image table.

If contact X in Figure 14.8 is OPEN when scanning resumes, register A is reset to 0 (all output bits OFF) *except* when Model 400 Revisions 1 and 2 are used with 16/32 function digital output modules. As described above, the value 32767 is retained (assuming that contact X was previously CLOSED) and output bits 1 through 15 are ON. Note that if contact X is CLOSED *when* scanning resumes, both the 4/8 and 16/32 function digital outputs behave the same.



Contact and Processor Condition	Output Registers 4/8	16/32
CLOSED before halt and OPEN when scanning resumes	0	32767
CLOSED before halt and CLOSED when scanning resumes	32767	32767

Figure 14.8 4/8 and 16/32 Function Digital Output Behavior in a LET Statement

To ensure that 16/32 function output registers are reset in Revisions 1 and 2, program a rung as illustrated in Figure 14.9:

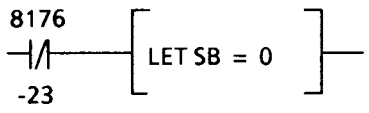


Figure 14.9 Rung Used for Model 400 Revisions 1 and 2 to Cause 4/8 and 16/32 Function Digital Outputs to Behave the SAME

In Figure 14.9, "B" represents the register(s) addressed to the 16/32 function output modules to be reset. The normally CLOSED (N.C.) 8176-23 contact is only closed during initialization and has no effect during subsequent scans. Matrix techniques may be used to reset blocks of consecutive registers.

14.6 Run-time Operational Checks

The Model 400 performs a number of internal diagnostic checks. Among these are:

1. A continuous check made to insure that the internal clock driving the processor logic is fully operational (hardware watchdog timeout).

2. Hardware circuitry halts the processor in the event the microprocessor fails to report every 2.5 milliseconds (software watchdog timeout).

3. Each word of user ladder and register memory incorporates a parity bit. The processor checks each examined word for proper parity. If an error is detected, the processor halts operation and turns external outputs off. (In this case, the HALT LED flashes and the MEMORY LED is illuminated.)

4. The control processor gives the scan processor a rung to execute at the end of each scan (a "fault rung"). If the rung is executed incorrectly, all outputs are turned OFF and an error is generated.