RUGID RTU TECHNICAL MANUAL

RUG6, RUG7, RUG8 MODELS

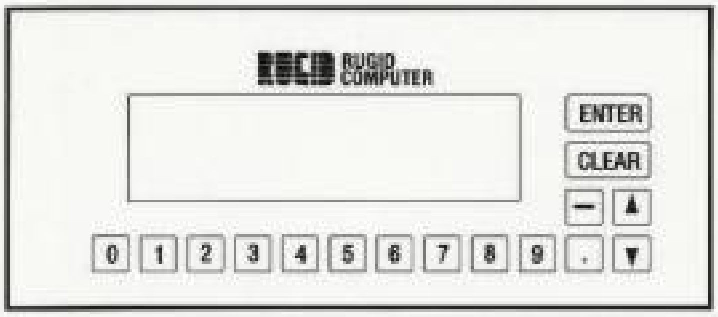


TABLE OF CONTENTS

TABLES

FIGURES

SECTION 1

1.0 INTRODUCTION

Congratulations on obtaining the most highly integrated, complete, and capable control and data acquisition microcomputer on the market today. The RUGID series of small computers will put you into any of a multitude of applications with ease of integration you never imagined. These computers are equipped with all the computing and peripheral capability necessary to perform most laboratory, municipal, field, and home control and data acquisition functions. The supplied software is integrated with the hardware to perform all real-time scanning functions, so all you have to do is define your application in a fairly high level sense and rely upon the RUGID system to handle the details.

Each RUGID computer is equipped with capabilities not found in most commercially available computers such as nonvolatile memory, watchdog timer, write protected memory, extended temperature range, CMOS logic (for good noise immunity and low power), brownout detector, auto boot system, fully secured array transfer communications system, and numerous control and data acquisition interfaces. Most of these are necessary for the majority of control applications, but are seldom present in any but the most expensive control computers. Often the user is required to integrate these capabilities into a computer not intended for control work, and then write software to enable the various pieces to work together. What results is a control system that usually does its job but intermittently fails and must be serviced. RUGID computers remove this risk by providing the necessary interfaces and security functions already integrated together in a unit that is ready to go to work when it arrives.

This manual will provide the necessary data to enable you to do most standard real-time control applications. With it you can set up the RUGID computer to, for example, control a fairly sophisticated laboratory process, perform production testing of electronic circuit boards, control a small water treatment plant, control the most sophisticated traffic intersection, acquire data at a remote field site and transmit back in response to a phone inquiry, or hook it up to an IBM, Apple or other computer to add data acquisition and control capability.

This manual presents a complete description of the BASIC instruction set with numerous examples, but by no means is it a BASIC programming manual that will provide sufficient tutorial to enable the novice programmer to conveniently learn to program in BASIC. For that, consult one of the many excellent manuals on BASIC programming.

Figures 1.1 and 1.2 present drawings showing the typical appearance of RUG6/8 and RUG7 units respectively.

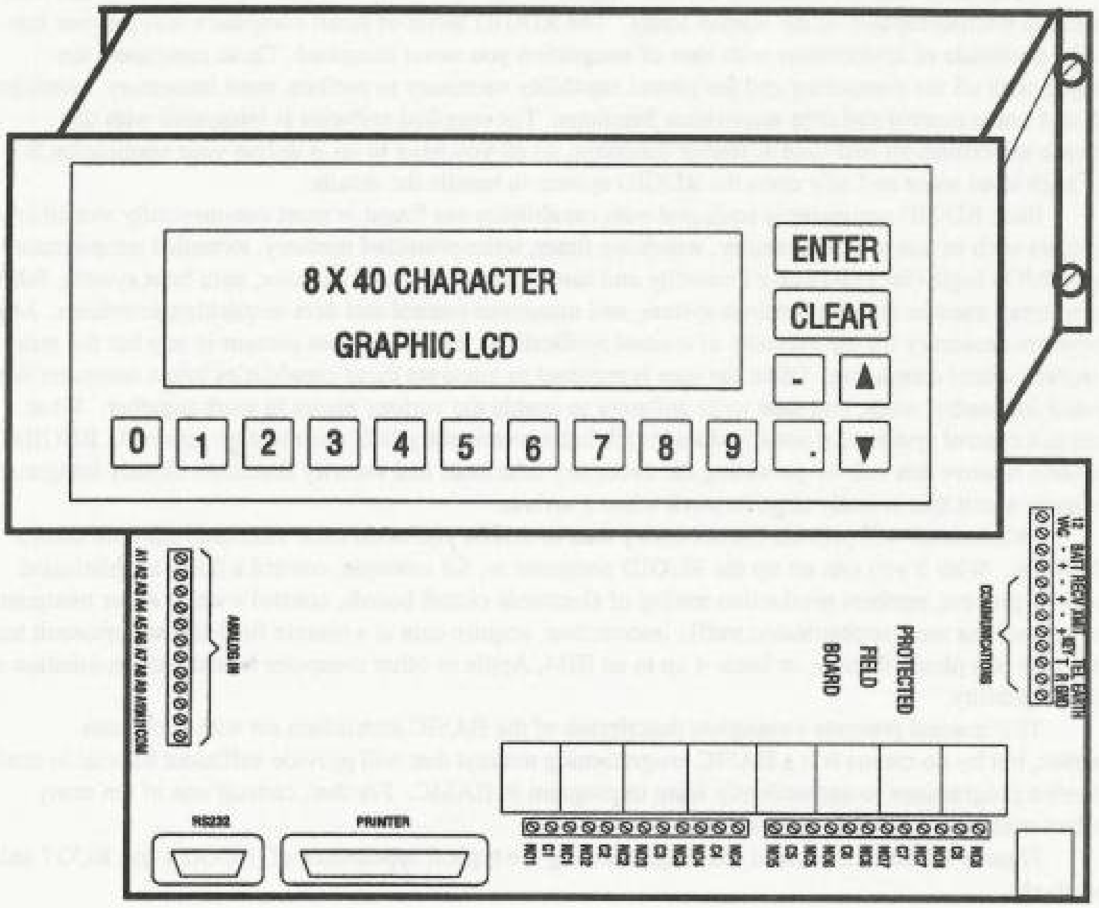


Figure 1.1 RUG6 or RUG8 SET UP FOR BACKPAN MOUNTING

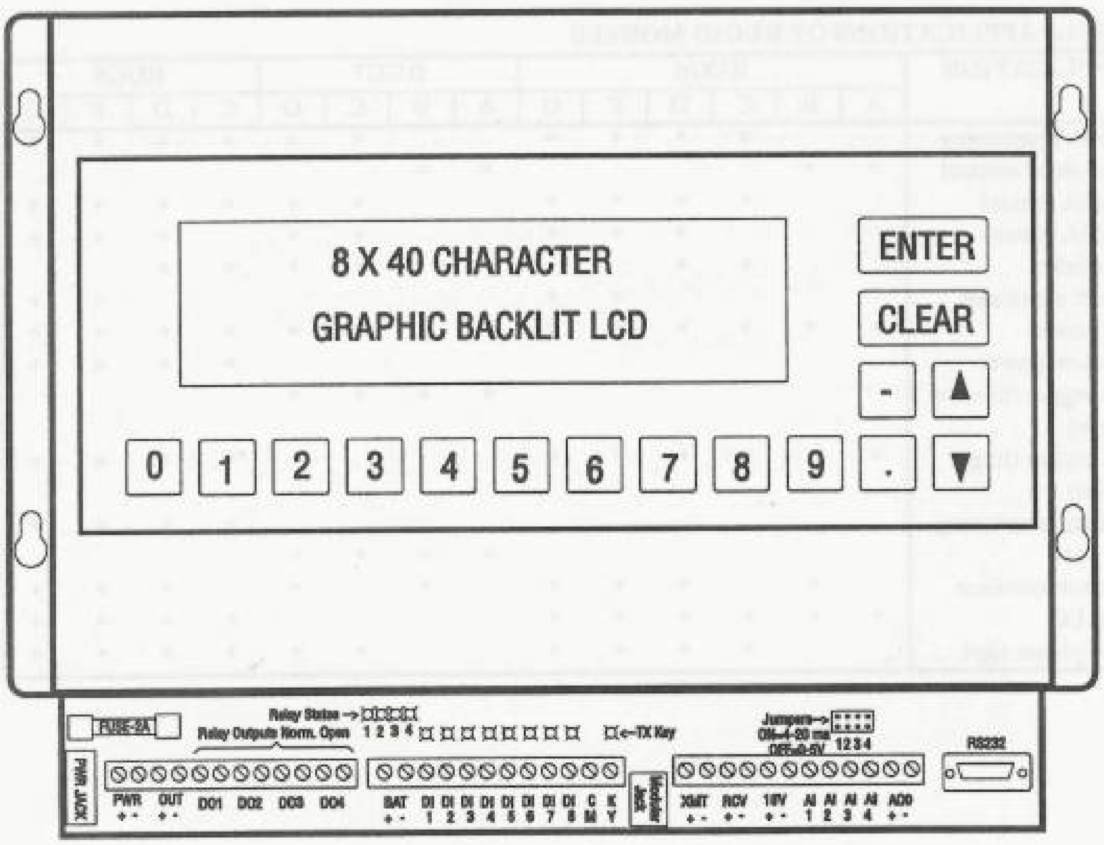


Figure 1.2 RUG7D

1.1 UNIT FEATURE SUMMARY

This manual covers three main RUGID Computer models, the RUG6, RUG7, and the RUG8. The main applications of these three models are summarized in Table 1.1. Differences between the models are summarized in Table 1.2, and consist mainly of base input/output (I/O) complement. Throughout the manual, you can assume that the features discussed apply to all models unless the text specifies that certain features apply only to certain models. All models use the same EPROM, so software upgrades can be obtained at any time simply by purchasing a more recent EPROM. All units program identically; and all units communicate identically. Therefore a system can be composed of a mix of different RUGID unit types to suit individual site I/O requirements. The following table helps identify the applications for which each model is best suited.

Table 1.1 APPLICATIONS OF RUGID MODELS

APPLICATION	RUG6						RUG7				RUG8			
	A	B	C	D	F	G	A	B	C	D	C	D	F	G
Remote monitoring			*	*	*	*			*	*	*	*	*	*
Stand alone control	*	*					*	*						
SCADA remote			*	*	*	*			*	*	*	*	*	*
SCADA master				*	*	*			*	*		*	*	*
Autodialer			*	*					*	*	*	*		
Speech autodialer					*	*							*	*
Low power	*	*	*	*	*	*	*	*	*	*	*	*	*	*
Very low power											*	*	*	*
Data logger (modest storage)							*	*	*	*				
Data logger (large storage)	*	*	*	*	*	*					*	*	*	*
Weather monitoring											*	*	*	*
Low cost							*	*	*	*				
Operator interface		*		*	*	*		*		*		*	*	*
Large I/O	*	*	*	*	*	*					*	*	*	*
Radio/phone appl.			*	*	*	*			*	*	*	*	*	*

TABLE 1.1 SUMMARY OF MODEL FEATURES

FEATURE	RUG6						RUG7				RUG8			
	A	B	C	D	F	G	A	B	C	D	C	D	F	G
BASIC in EPROM	*	*	*	*	*	*	*	*	*	*	*	*	*	*
Battery backed RAM	*	*	*	*	*	*	*	*	*	*	*	*	*	*
Watchdog timer	*	*	*	*	*	*	*	*	*	*	*	*	*	*
Clock/Calendar	*	*	*	*	*	*	*	*	*	*	*	*	*	*
Loop supply	*	*	*	*	*	*	*	*	*	*	*	*	*	*
Battery charger	*	*	*	*	*	*	*	*	*	*				
Reference V output											*	*	*	*
LCD/Keyboard		*		*	*	*		*		*		*	*	*
RS232 Port 1	*	*	*	*	*	*	*	*	*	*	*	*	*	*
Printer Port	*	*	*	*	*	*					*	*	*	*
300/1200 Modem			*	*	*	*			*	*	*	*	*	*
Touchtone TX/RX			*	*	*	*			*	*	*	*	*	*
Radio/Tel 4-wire			*	*	*	*			*	*	*	*	*	*
Tel 2-wire			*	*	*	*			*	*	*	*	*	*
RS232 Port 2											opt	opt	opt	opt
ALERT Tone set			opt	opt	opt	opt			opt	opt	opt	opt	opt	opt
CRC secured com			*	*	*	*			*	*	*	*	*	*
Store & forward			*	*	*	*			*	*	*	*	*	*
MODBUS on Port 1	*	*	*	*	*	*	*	*	*	*	*	*	*	*
Speech synth, min.					1	4							1	4
SLEEP mode							opt	opt	opt	opt	*	*	*	*
Base I/O:														
Analog inputs	11	11	11	11	11	11	4	4	4	4	11	11	11	11
Digital inputs	16	16	16	16	16	16	8	8	8	8	8	8	8	8
Relay outputs	8	8	8	8	8	8	4	4	4	4	4	4	4	4
Analog outputs	0	0	0	0	0	0	0	0	0	0	0	0	0	0
User adj. gain amp											3	3	3	3
Anemometer input											1	1	1	1
Expansion I/O #:														
AI, opto isolated	64	64	64	64	64	64					64	64	64	64
Digital in	128	128	128	128	128	128					128	128	128	128
Digital out	128	128	128	128	128	128					128	128	128	128
AO, opto isolated	32	32	32	32	32	32	1	1	1	1	32	32	32	32
Other expansion:														
RAM bank, 640K	*	*	*	*	*	*					*	*	*	*
Normal current, ma.	46	68	112	134	172	172	51	73	110	132	112	134	172	172
SLEEP current, ua.							93	93	93	93	93	93	93	93

Expansion capability is in addition to base I/O. Number of channels for I/O represents addressing limitations; the card cage space could limit I/O expansion to fewer channels.

1.2 UNIT BLOCK DIAGRAMS

Figures 1.2.1, 1.2.2 and 1.2.3 present the unit functional block diagrams.

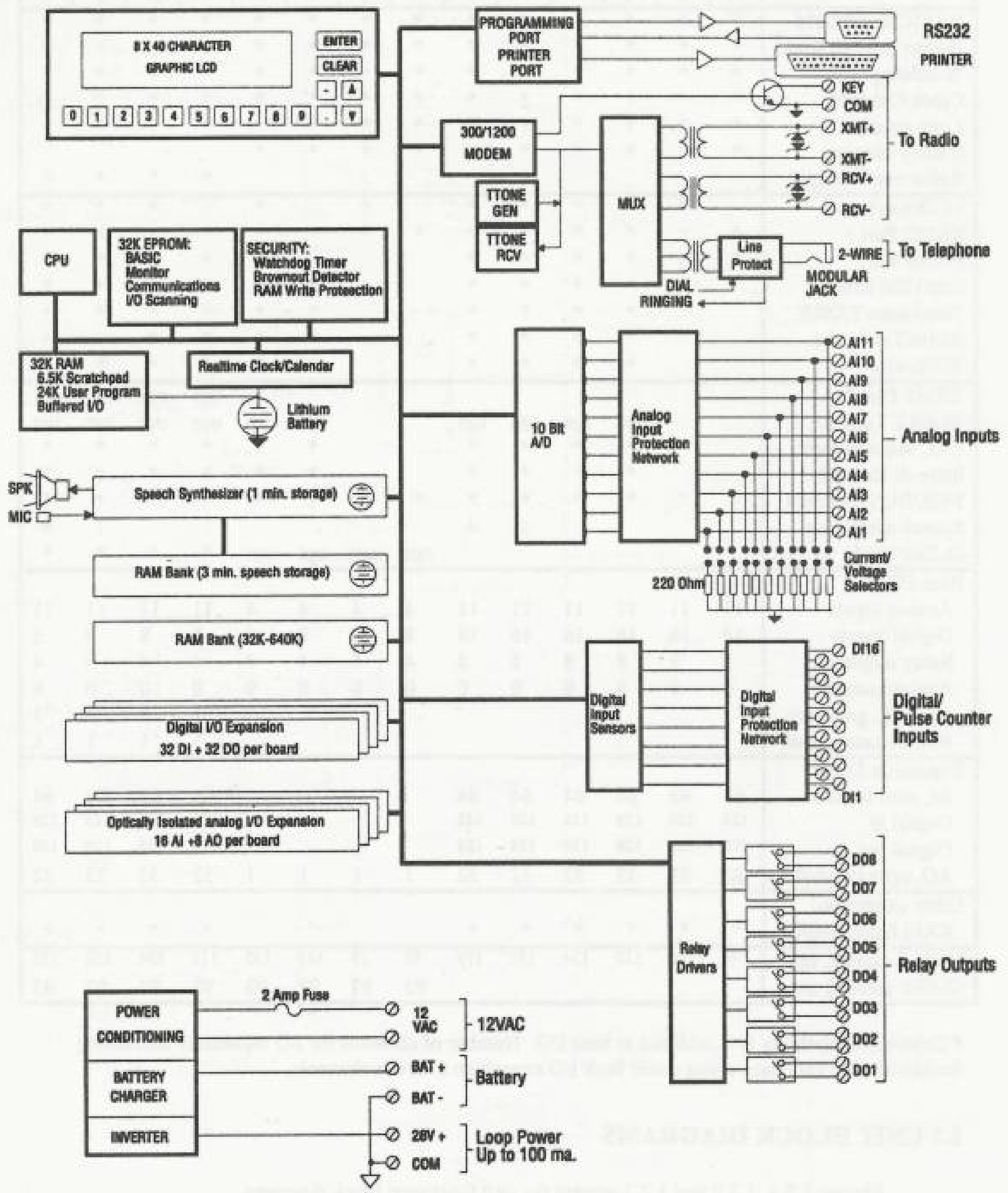

8 X 40 CHARACTER GRAPHIC LCD
ENTER
CLEAR
0 1 2 3 4 5 6 7 8 9 .
PROGRAMMING PORT PRINTER PORT
RS232
PRINTER
KEY
COM
XMT+
XMT-
RCV+
RCV-
To Radio
300/1200 MODEM
TTONE GEN
TTONE RCV
MUX
Line Protect
2-WIRE
To Telephone
MODULAR JACK
DIAL RINGING
CPU
32K EPROM: BASIC Monitor Communications I/O Scanning
SECURITY: Watchdog Timer Brownout Detector RAM Write Protection
32K RAM 6.5K Scratchpad 24K User Program Buffered I/O
Realtime Clock/Calendar
Lithium Battery
SPK
MIC
Speech Synthesizer (1 min. storage)
RAM Bank (3 min. speech storage)
RAM Bank (32K-640K)
Digital I/O Expansion 32 DI + 32 DO per board
Optically Isolated analog I/O Expansion 16 AI +8 AO per board
10 Bit A/D
Analog Input Protection Network
AI11
AI10
AI9
AI8
AI7
AI6
AI5
AI4
AI3
AI2
AI1
Analog Inputs
Current/ Voltage Selectors
220 Ohm
Digital Input Sensors
Digital Input Protection Network
DI16
DI1
Digital/ Pulse Counter Inputs
Relay Drivers
DO8
DO7
DO6
DO5
DO4
DO3
DO2
DO1
Relay Outputs
POWER CONDITIONING
BATTERY CHARGER
INVERTER
2 Amp Fuse
12 VAC
12VAC
BAT +
BAT -
Battery
28V +
COM
Loop Power Up to 100 ma.

Figure 1.2.1 RUG6 BLOCK DIAGRAM

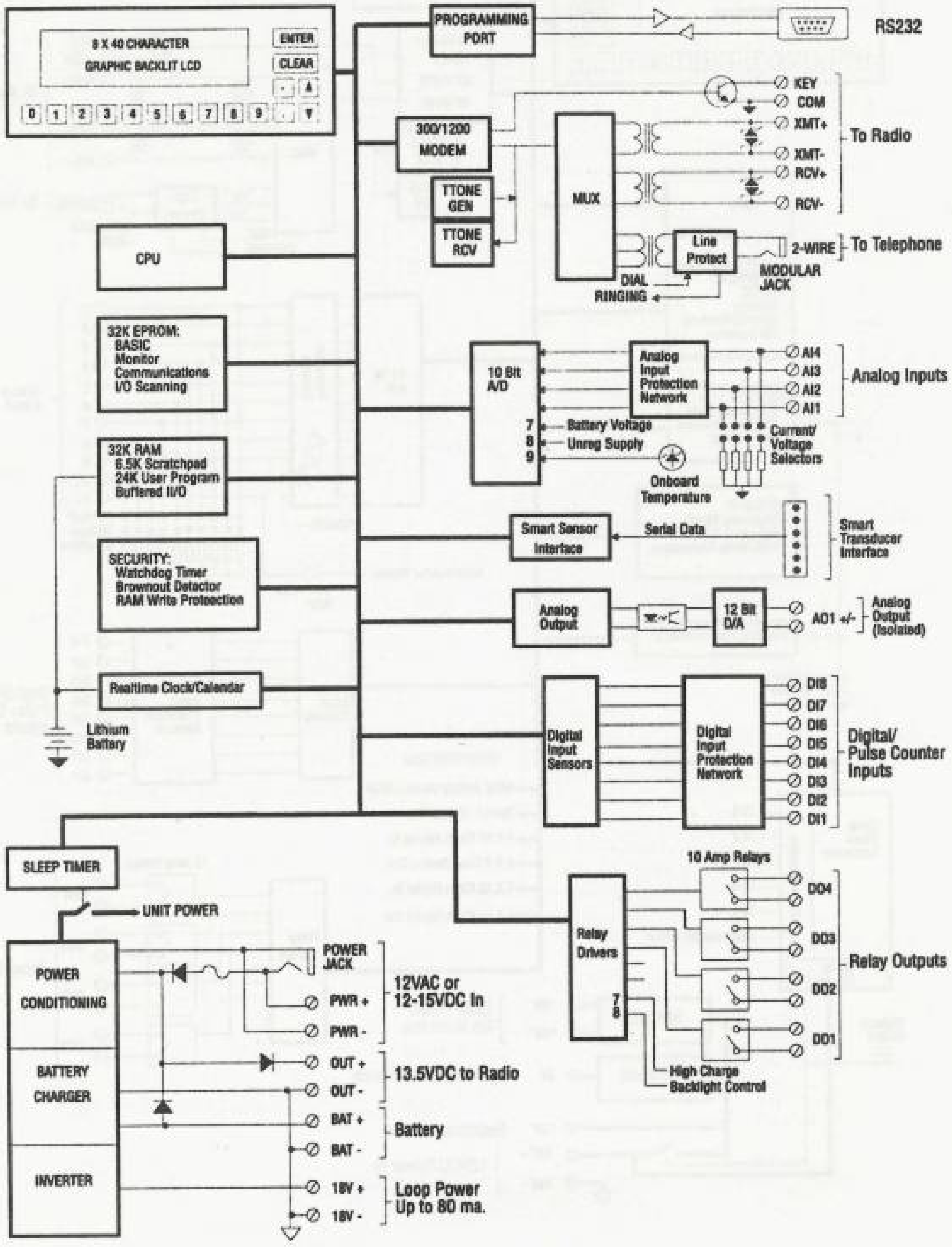


Figure 1.2.2 RUG7 BLOCK DIAGRAM

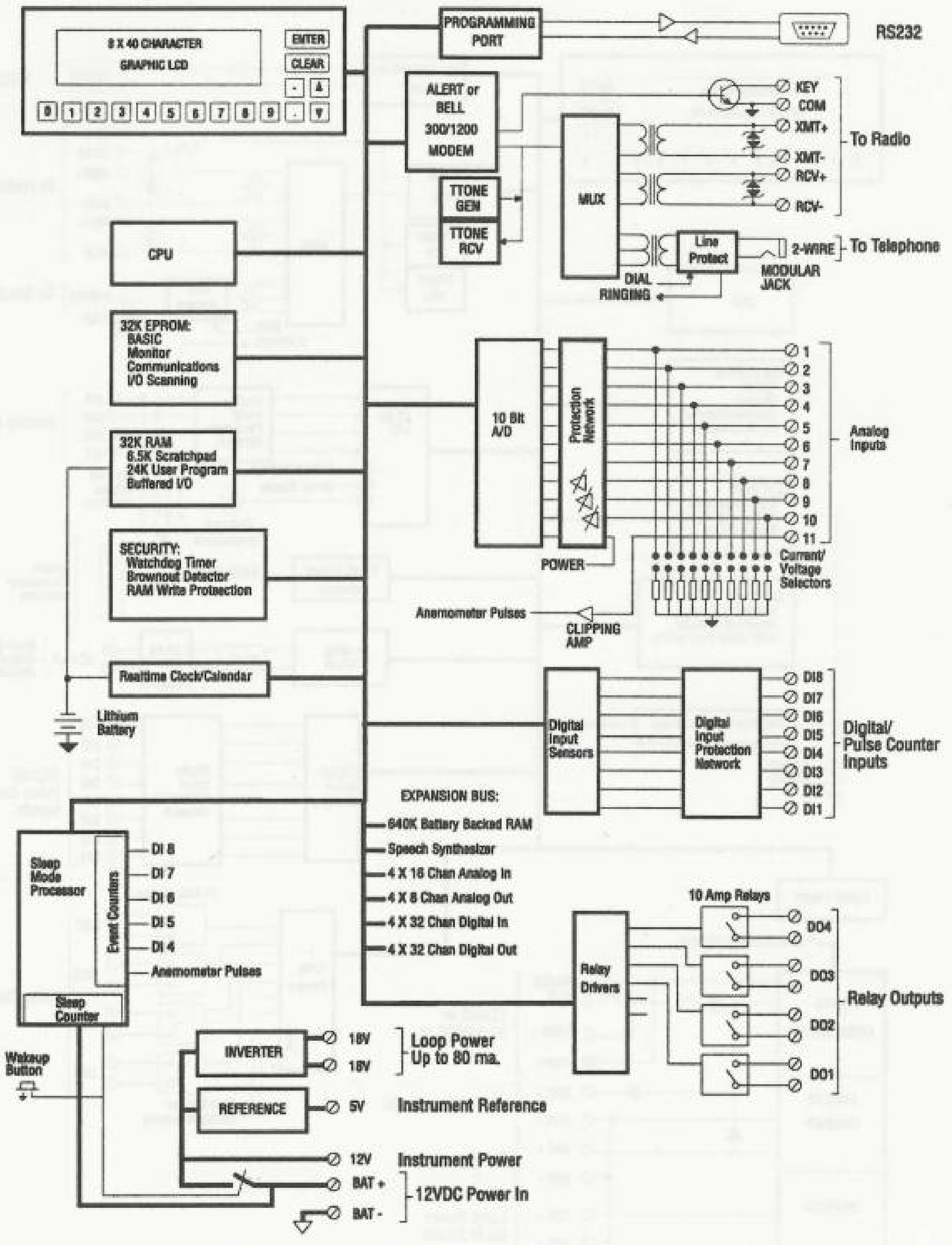


Figure 1.2.3 RUG8 BLOCK DIAGRAM

1.3 CIRCUIT BOARD FUNCTIONS

A RUGID unit can consist of a minimum of just the main board, or the main board plus many combinations of expansion boards, display and keyboard. This section summarizes the functions of the expansion boards so that you will know what is available if you wish to expand your RUGID unit.

1.3.1 Main Board

(can be used stand alone for imbedded applications)

- Processor
- 32K EPROM with BASIC
- Brownout detector
- LCD interface
- Negative voltage inverter
- Parallel printer port
- 11 analog inputs
- 32K Battery backed RAM
- Watchdog timer
- Real-time clock/calendar
- Keyboard interface
- RS232 port #1
- 16 digital inputs
- 8 relay driver outputs

1.3.2 RAM Bank

(can use one per unit for data storage, plus one per unit to expand speech storage)

- 32K to 640K bytes RAM, battery backed up

1.3.3 Speech Synthesizer

- ADPCM speech digitizer
- 1 minute nonvolatile speech storage
- ADPCM speech synthesizer
- Audio amplifier
- Interface to modem board

1.3.4 Analog I/O Expansion

(up to four boards per unit max)

- 16 analog inputs, 12 bits, optically isolated, 4-20 ma.
- 8 analog outputs, 12 bits, optically isolated, 4-20 ma. or voltage

1.3.5 Digital I/O Expansion

(up to four boards per unit max)

- 32 digital inputs, resistor isolated
- 32 relay driver outputs
- Relay back EMF protection

1.3.6 Main Unit Protected Field Board

- 84 screw terminals
- Analog input current sensors
- Analog input current loop/voltage compatibility selectors
- Digital input and digital output LED's
- Digital input resistive isolators
- Eight 10 amp relays
- Power fuse, rectification, filtering
- RS232 (DB9) and parallel printer (DB25) connections

- 100 ma inverter type loop power supply
- 140 ma temperature compensated battery charger
- IEEE surge protection on all I/O
- clamping zeners and current limiting resistors per analog input channel

1.3.7 Modem/Communications Board

- 300/1200 baud modem
- Auto dialer
- Auto answer
- 2-wire/4-wire interface, software selected
- RS232/RS422 software selected
- Speech synthesizer interface
- Touchtone detector
- Transmitter keyer

1.3.8 Digital I/O Expansion Field Wiring Board

- 48 screw terminals with LED's

1.3.9 Analog Expansion Field Wiring Board

- 32 screw terminals

1.3.10 Relay Field Board

Board with 8 ten amp form C relays, relay drivers, LED's and screw terminals. Board requires separate 12 VDC for operation. Boards can be daisy chained up to four in a row for a total of 32 relays attached to one MIO or DIO board internal to the RUG6 or RUG8

1.3.11 Optical Field board

Board with 32 channels of optically isolated digital inputs. Board requires external 12 VDC to maintain full isolation.

1.3.12 Communication board (RUG8 only)

Circuit board that mounts internal to the RUG8 unit to provide the following:

- Modem, Bell 103/212
- Modem with ALERT tones (optional), software selectable
- Touchtone generation and detection
- RS232 port (optional)
- Onboard low power SLEEP mode processor to trigger wake up after time delay
- Onboard low power processor to count external events and interface to shaft encoders.

1.3.13 Low power field board (RUG8 only)

Field board included with all RUG8's with the following interfaces:

11 analog inputs
8 digital inputs
4 relay outputs
3 user adjustable gain analog inputs (included in 11 total)

1 anemometer input for counting anemometer pulses directly
Loop power supply
4.7 V DC reference for interfacing to potentiometers directly
SLEEP mode latch and wakeup button
SLEEP mode power switch

1.3.14 Phone line MUX amplifier

Amplifier/isolator/summing amplifier board for interfacing multiple phone lines to a single RUGID. The board provides 10db transmit gain and transformer isolation. Several MUX boards can be daisy chained together at a site to sum multiple phone lines, providing transmit gain for each channel, summing the received signals and maintaining 600 impedance throughout.

1.4 BOARD CURRENT CONSUMPTION

The following circuit board power consumptions were measured at room temperature using a 12 VDC power supply, on a unit running the diagnostic program DIAGNOS6. These values can vary substantially depending upon the proportion of time spent accessing the individual boards. The unit was observed to begin operation with a DC input of 9.1 volts, and to drop out, due to the brownout detector, at 8.1 VDC.

BOARD	CURRENT, ma.	
CPU	46	
CPU+LCD	68	
Modem	20	
RAM bank, 128K11		
Analog I/O	25	
Digital I/O	20	
Speech	28	
Protected field board	20	no I/O on
	3	per digital input turned on (due to LED)
	35	per relay turned on (due to relay+LED)
	2X	the loop current being sourced
Analog field board	0	
Digital field board	3	per channel turned on (due to LED)
Optical field board	0	
Communication board	20	
Low power field board	20	no I/O on
	3	per digital input turned on (due to LED)
	35	per relay turned on (due to relay+LED)
	2X	the loop current being sourced

RUG7:

RUG7A	66	
RUG7B	88	
RUG7C	86	
RUG7D	106	
Backlight on, add	50	
Each relay, add	35	
Loop power supply, add	2X	the loop current being sourced

SECTION 2

2.0 GETTING STARTED

In order to use your RUGID computer you will need 12 VAC or DC power. If you have ordered a preprogrammed unit, follow the inclosed instructions regarding setup and operation. If your unit is unprogrammed, it will have the DIAGNOSX.CMP program installed, where X is your unit model type (6, 7 or 8) which is installed to enable you to observe unit operation and which is used during our automated test before shipment. You will need to hook up a computer to communicate with RUGID and load a program.

2.1 APPLYING POWER

All RUGID units run on nominal 12 VDC or 12VAC power. Integral diode isolation protects the units from voltage reversal, and prevents the unit from providing current to a DC source when an AC voltage source is present. Refer to figure 2.1.1 for proper voltage source connections.

2.2 CONNECTING A COMPUTER

Using a computer to enter and maintain RUGID programs is necessary since the computer provides you word processing and mass storage not available in the RUGID unit. Figure 2.2.1 illustrates the proper connections to the main board serial port. A DB9 female to DB9 male cable is required to connect a PC to the RUGID's programming port. You will need a terminal emulation program for interacting with RUGID and for transferring programs to RUGID. Programs such as the public domain QMODEM will work fine. Make sure that serial port parameters are set to 9600 baud, 8 bit word, one stop bit and disabled parity for initial communications.

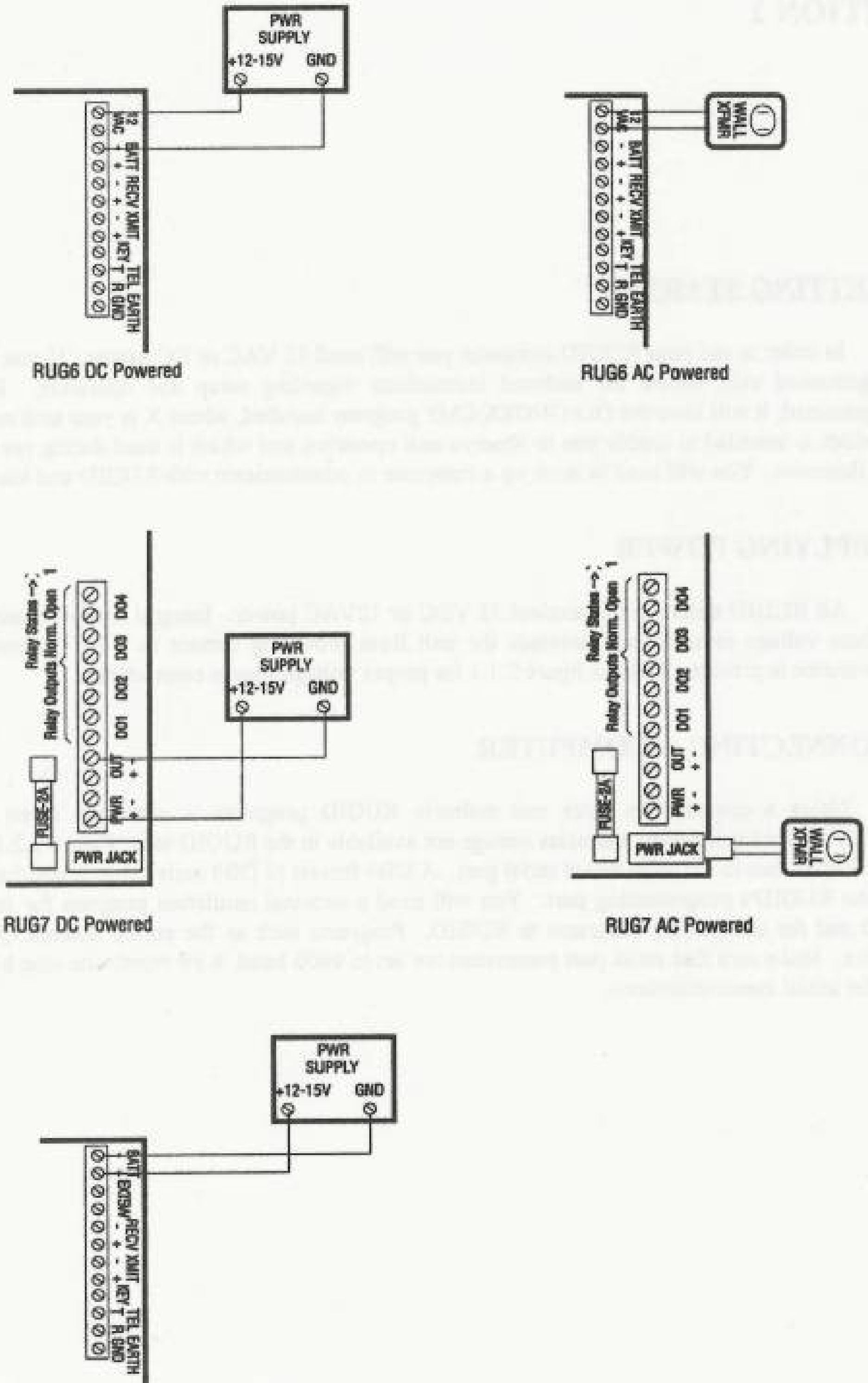


FIGURE 2.1.1 POWER CONNECTION

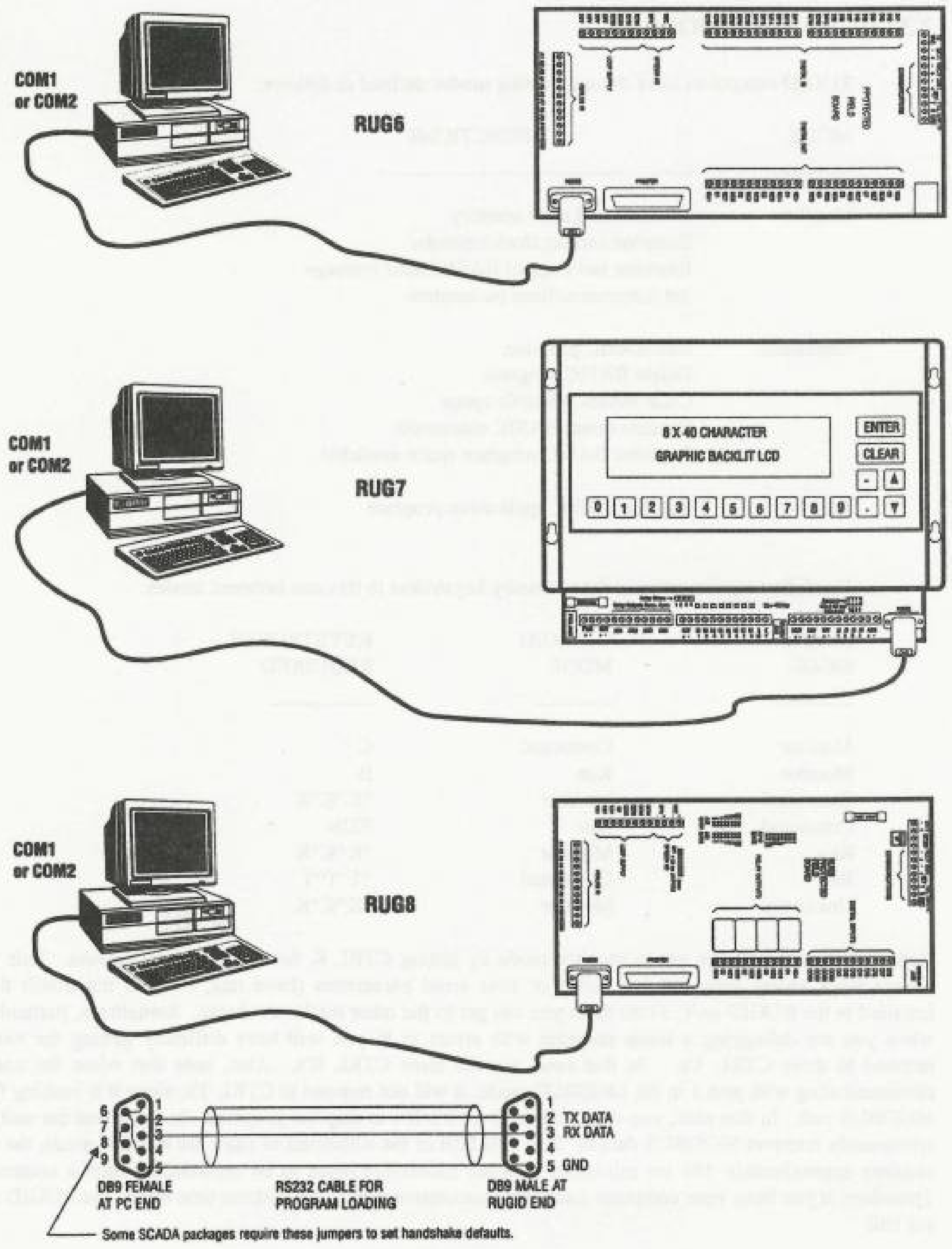


FIGURE 2.2.1 CONNECTION TO IBM PC COMPATIBLE COMPUTER

2.3 OPERATING MODES

RUGID computers have three operating modes defined as follows:

MODE	FUNCTIONS
Monitor	Examine and alter memory Examine and set clock/calendar Examine last trapped BASIC error message Set communications parameters
Command	Edit BASIC program Delete BASIC program Clear BASIC variable space Execute direct BASIC statements Examine BASIC program space available
Run	Run the BASIC application program

The following list presents the necessary keystrokes to traverse between modes:

CURRENT MODE	DESIRED MODE	KEYSTROKES REQUIRED
Monitor	Command	C
Monitor	Run	B
Command	Monitor	^K^K^K
Command	Run	RUN
Run	Monitor	^K^K^K
Run	Command	^T^T^T
Unknown	Monitor	^K^K^K

Note that you can always get to monitor mode by hitting CTRL K three times in succession. This will always work unless your unit has failed or your serial parameters (baud rate, etc.) do not match those installed in the RUGID unit. From there you can get to the other modes as shown. Sometimes, particularly when you are debugging a Basic program with errors in it, you will have difficulty getting the unit to respond to three CTRL T's. In that case, use the three CTRL K's. Also, note that when the unit is communicating with port 1 in the MODBUS mode, it will not respond to CTRL T's, since it is looking for a MODBUS poll. In that case, you must use the 3 CTRL K's to stop the program. In order that the unit not erroneously interpret MODBUS data of $0B $0B $0B as the command to enter the monitor mode, the unit requires approximately 100 ms minimum between successive bytes to be regarded as such a command. Therefore, if you have your computer issue three successive CTRL K's without time delay, the RUGID will not halt.

2.4 POWER UP INITIALIZATION

Upon power up or watchdog timer reset of the processor, the unit will examine an array of pointers and other data necessary for unit operation. These data are held triply redundantly in nonvolatile memory and voted upon to ascertain that they are intact. The data are the following:

LOCATION	VALUE	FUNCTION
$7F00	$00	Puts unit in monitor mode.
$7F01	$01	Modem UART command: disabled parity
$7F02	$18	Modem UART control: 8 bits, 1 stop, 1200 baud
$7F03	$00	Unit address
$7F04	$01	Term. UART command: disabled parity
$7F05	$1E	Term. UART control: 8 bits, 1 stop, 9600 baud
$7F06,7	Varies	Reset vector to RUGID entry point
$7F08,9	Varies	Interrupt vector to RUGID interrupt entry pt.
$7F0A	$03	Auto-answer on 3 rings, modem in answer mode
$7F0B	$00	CRC off, diagnostics off, dialup line
$7F0C	$10	CRC transmit delay 289 ms.
$7F0D,E,F	$00	Future

Two entries for any item must match or the default values will be installed in place of those present for that item. Full definition of the above parameters is contained in Appendix D.

Once voting is satisfied, the unit tests PROM contents for correctness and, if not correct, issues an error message if it can. If PROM contents are correct, the processor will jump to either the monitor, BASIC command mode, or will run the BASIC program depending upon the operating mode in effect before the power outage or watchdog timer reset event.

2.5 USING QMODEM TO COMMUNICATE WITH RUGID

QMODEM is a public domain IBM compatible shareware program useful for a variety of communications purposes, many of which we have not tested. Other programs, such as PROCOMM, CROSSTALK, Windows Terminal, Microsoft Work's terminal mode and others will work just as well for communicating, and loading programs. This section discusses the use of QMODEM to interact with RUGID and to transfer programs to RUGID. Before invoking QMODEM, you should connect your computer's serial port to RUGID's programmin port using a DB9 female to DB9 male cable. Also be sure you know RUGID's communications parameters (baud rate, word length, parity and number of stop bits) before invoking QMODEM. QMODEM has limited parameter setting options in that 8 bit words can only be used with disabled parity. We recommend that you use the parameter setting 9600,N,8,1 which means 9600 baud, no parity, 8 bit word, 1 stop bit. QMODEM generally starts with this parameter set.

2.5.1 INVOKING QMODEM

Before invoking QMODEM, make sure the QMODEM files are in your default drive, usually drive A for floppy systems or drive C for hard disk systems. Also, hit the CAPS-LOCK key so that all your communications with RUGID are in upper case. Type in QMODEM(CR) to start QMODEM. The (CR) is the carriage return or ENTER key on your keyboard. QMODEM starts in a terminal emulation mode that will transfer everything you type to RUGID and will display RUGID's replies to your CRT. To see if you have communications, hold the control key down (CTRL) and hit the 'K' key three times (ctrl-KKK). This will reset RUGID to the monitor mode assuming you have correct connections and correct communications parameters. If you get no response, follow the procedure below for setting the IBM's baud rate. If you still have problems, check your cabling. If that doesn't work, call RUGID.

2.5.2 SETTING QMODEM'S BAUD RATE, WORD LENGTH,...

1) Hit ALT-P. QMODEM will present parameter options.

2) Select new parameters, e.g., typing 'C' will select 300 baud, no parity, 8 bit word length and one stop bit.

3) QMODEM will transfer back to terminal emulation mode whereby your keystrokes go directly to RUGID and RUGID's responses appear on your CRT.

Note: If you wish for QMODEM to boot up with a particular baud rate, word length, etc., run the QINSTALL program which sets up QMODEM's initialization and lets you modify the parameters.

2.5.3 LOADING A PROGRAM INTO RUGID

After you have confirmed communications with RUGID, you can load programs from disk to RUGID using this procedure:

1) Put RUGID in command mode, i.e., the mode in which you can edit BASIC programs. If RUGID is running a program, type CTRL- T three times to halt the program. RUGID should respond with 'BREAK IN 1234 OK', indicating the line that was interrupted. If that doesn't work, type three control K's and, after the "WELCOME TO RUGID MONITOR..." prompt, type 'C' to transfer to command mode. RUGID should respond with the 'OK' prompt.

2) Type 'NEW' to delete the old program. You should get the 'OK' prompt.

3) Hit the 'Pg Up' key. This will give you QMODEM's upload menu.

4) Choose option '1' for an ASCII transfer.

5) Enter the file name you wish to transfer, e.g., 'TESTPGM.CMP'.

6) When asked for the transfer mode select '1', prompted.

7) When asked for a prompt, hit the RETURN (ENTER) key. At this point you should see your program scroll up the screen line by line. This is actually the program being echoed by the RUGID. If you see error messages, you may have asked QMODEM to load a text source file instead of one of the files with ".CMP" after the file name. If so, press the PGUP key to terminate the program loading process and go back to step 2.

8) After the transfer is complete, QMODEM will put you back in terminal mode wherein your keystrokes go directly to RUGID and RUGID's responses appear on your CRT. If you want to confirm that your program transferred, type 'LIST' to have RUGID list the program to your CRT. You will have to hit the space bar after the first line and every 20 lines thereafter. RUGID does this to keep the program from scrolling off the screen before you can read it.

9) RUGID will attempt to use the variables that were present when you stopped your previously running program. This will cause index errors if your new program is using array names with the same variable name but different index ranges. If this is the case, you must type "CLEAR" to clear out all old variables before you attempt to run the new program. Otherwise, the RUGID will use the variables present before the new program was loaded.

10) You may now type in 'RUN' to run the program.

11) If you wish to stop the program, type CTRL-T three times and wait a few seconds.

2.6 CONNECTING A PRINTER (RUG6 and RUG8 only)

The protected field board has a DB25 parallel printer port which is compatible with standard IBM PC type parallel printer cables. Any printer having a Centronics or parallel interface can be operated with the proper cable from this port. The cable is available from RUGID.

SECTION 3

3.0 CONNECTING AND CONTROLLING I/O

Connecting I/O properly is the key to obtaining accurate, reliable operation from the RUGID system. This section presents I/O schematics and example I/O hookups to aid in properly connecting I/O devices such as motor starters, transducers, control switches, etc. In general, the following guidelines will assist you in obtaining success in I/O interfacing:

1. Use twisted, shielded pair wires for low level signals (analog in, digital in, analog out).
2. Ground shields of input channels to RUGID's COMMON terminals only.
3. For analog signals, use current loops rather than voltage measurements.
4. Keep high current signals away from analog signals.
5. Keep AC or relay coil signals away from low level signals (analog inputs or digital inputs).

Refer to Figures 3.0A, 3.0B, and 3.0C for field board base I/O pinouts for RUG6, RUG7 and RUG8 respectively.

3.1 ANALOG INPUTS

RUGID models have varying complements of analog inputs as listed in the table below:

TABLE 3.1 BASE ANALOG INPUTS

CHANNEL	RUG6	RUG7	RUG8
AI%(1)	0-5 V or 4-20 ma.	0-5V or 4-20 ma.	0-5V or 4-20 ma.
AI%(2)	0-5 V or 4-20 ma.	0-5 V or 4-20 ma.	0-5 V or 4-20 ma.
AI%(3)	0-5 V or 4-20 ma.	0-5 V or 4-20 ma.	0-5 V or 4-20 ma.
AI%(4)	0-5 V or 4-20 ma.	0-5 V or 4-20 ma.	0-5 V or 4-20 ma.
AI%(5)	0-5 V or 4-20 ma.		0-5 V or 4-20 ma.
AI%(6)	0-5 V or 4-20 ma.		0-5 V or 4-20 ma.
AI%(7)	0-5 V or 4-20 ma.	Unregulated power	0-5 V or 4-20 ma.
AI%(8)	0-5 V or 4-20 ma.	Battery voltage	Adjustable gain
AI%(9)	0-5 V or 4-20 ma.	Onboard temperature	Adjustable gain
AI%(10)	0-5 V or 4-20 ma.		Adjustable gain
AI%(11)	0-5 V or 4-20 ma.		Anemometer input *

* The nominal 20 mv anemometer signal is connected to AI(11) on the RUG8 terminal board where it is amplified and clipped and then routed to digital input 8 where it can be counted in counters CO%(5) LS and CO%(13) MS. When AI%(11) is read, BASIC actually receives the unit's unregulated power voltage.

RUG7 units have no analog input expansion capability beyond the analog inputs listed above. RUG6 and RUG8 models can accept up to 4 analog input expansion boards, each having 16 optically isolated analog inputs.

3.1.1 BASE ANALOG INPUTS

The base analog input complement utilizes a 10 bit successive approximation A/D converter capable of up to 2000 samples per second. It includes sample and hold and channel multiplexing functions. For most channels identified in the table above, measurement range is nominally 0 to 5 VDC with the channel shorting bars removed, or 4-20 ma. current loop compatible with the channel shorting bars in place.

Background software samples the inputs and stores the results in the BASIC AI%() array. Channel 1 is stored in AI%(1); channel 2 in AI%(2), etc. The number of channels sampled is established by the dimension of the AI%() array. Entries in the AI%() array are in the form of raw count integers in the range of 0 to 1023. For a channel set up as a voltage input, an input of 0 volts should result in a raw count of 0; an input of 5 volts should result in a raw count in the range of 900 to 1023. Similarly, for a channel set up as a current input, currents of 4 ma and 20 ma should result in AI%() raw counts of approximately 650 and 3700 respectively. Note that these values can vary widely from unit to unit since the A/D reference is 5% accurate. Therefore, you must calibrate each input before accurate measurements can be obtained. Routines for handling calibration are listed in section 3.3 below.

Interrupt software samples the A/D as often as 225 times per second. On each access, zero to 10 samples can be taken. AI%(0) is used to tell background software the sample rate and number of samples to take on each cycle. The 2 bytes of AI%(0) are used as indicated below:

BITS	MEANING
0-7	Number of 225ths of a second between cycles
8-11	Number of samples to take each cycle
12-15	Type of A/D converter (use 0001)

To set the sample rate, the following relationship can be used:

SS=samples per second per channel
NC=number of channels to be sampled (1 to 11)
AI%(0)=4352+int(225/(SS*NC))

Some useful values for AI%(0) are:

$1114=4372	One sample per second per channel (recommended)
$1102=4354	10 samples per second per channel

Therefore, the following statements will give you the indicated analog input sample rates:

```
DIM AI%(8)
AI%(0)=4372          '1 sample per second per channel
AI%(0)=4354          '10 samples per second per channel
AI%(0)=0             'Stops analog input sampling
```

Note that time spent sampling analog inputs slows other processes, so sampling should be kept as slow as possible and still meet the requirements of the application. Figure 3.1.1 presents the analog input schematic for the main board in combination with the protected field board.

3.1.1.1 Base RUG7 Analog Inputs

As identified in Table 3.1.1, above, the RUG7 has four general purpose analog inputs (AI's 1 through 4) that can be set for 0-5V or 4-20 ma measurements using the channel shorting bars. In addition, the RUG7 uses analog inputs 7, 8 and 9 for internal measurements. The code segments below will give battery voltage, bus voltage and onboard temperature within 5%.

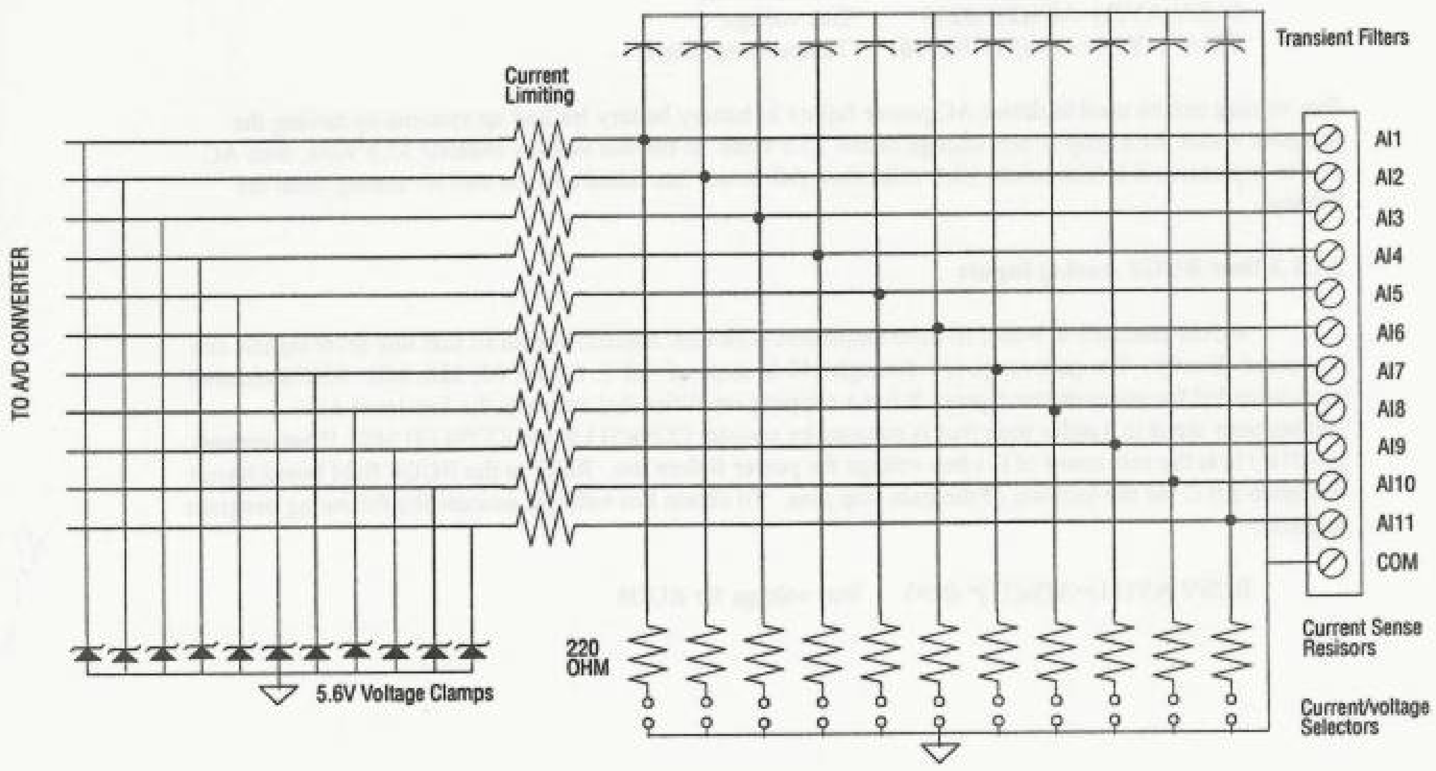
```
BATTV AY(7)=AI%(7)*.0245        'Battery voltage
BUSV AY(8)=AI%(8)*.0245         'Bus voltage
TEMP AY(9)=AI%(9)*.742-381      'Temperature, deg F
```

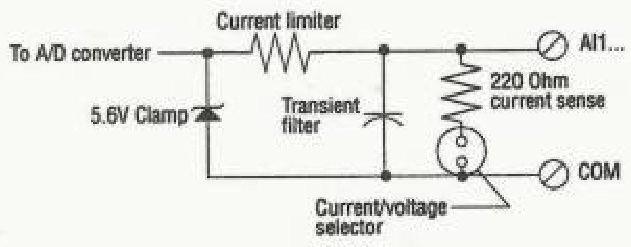
Bus voltage can be used to detect AC power failure in battery battery backed up systems by having the program watch for a drop in bus voltage below 13.5 volts. If the bus voltage exceeds 13.5 volts, then AC power is present; if it falls below 13.5 volts, then AC power has failed and the unit is running from the battery.

3.1.1.2 Base RUG8 Analog Inputs

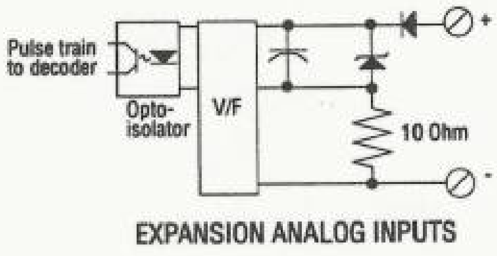
RUG8 channels 8, 9 and 10 have amplifiers with user adjustable gain so that low level signals can be sensed directly. The gain range is 1 through 240 in steps of 1,2 5, 10, 20, 60, 120, 240. RUG8 channel 11 is intended for anemometer inputs. It has a clipping amplifier that converts the low level AC anemometer input to a pulse train that is counted by counter CO%(5) LS and CO%(13) MS. What appears in AI%(11) is the raw count of the bus voltage for power failure use. Refer to the RUG8 field board layout in Figure 3.0 C for the location of the gain step pins. To obtain bus voltage, execute the following program segment:

```
BUSV AY(11)=AI%(11)*.0293       'Bus voltage for RUG8
```





BASE ANALOG INPUTS



EXPANSION ANALOG INPUTS

FIGURE 3.1.1A ANALOG INPUT SCHEMATIC

3.1.2 EXPANSION BOARD ANALOG INPUTS

Each expansion board has 16 optically isolated analog inputs. Twelve bit resolution is obtained using a precision voltage to frequency technique employing pulse transmission across an optical isolator for each channel. Each channel is isolated from the other channels and from the processor by 2500 VDC. The channels are compatible with the 4 to 20 ma. standard and require a loop current of at least 3 ma. to operate. Circuitry on the field side of the isolator is loop powered. Background software samples the channels as fast as possible and stores the samples in the BASIC array AI%() beginning with AI%(12). IMPORTANT: Expansion board analog inputs are stored as the INVERSE of the current value on the analog input. Therefore, you must perform a calculation of the form Y=1/AI%() to obtain a raw count proportional to the analog input current. The software in section 3.3 takes care of this for you. The expansion board provides greater accuracy than the main board converter. Figures 3.1.2A and 3.1.2B provide I/O schematics and example connections respectively.

3.2 ANALOG OUTPUTS

There are no analog outputs included in the base RUGID units. A single analog output channel is optional on the RUG7 model. Up to 4 analog output boards can be installed in either the RUG6 or RUG8 models. Each analog I/O expansion board has 8 optically isolated analog outputs with voltage and current loop drivers and 12 bit resolution. The optical isolators provide 2500 VDC channel to channel and channel to computer isolation. Analog outputs are controlled by writing to the AO%() array. The allowed range of AO%() is 1 through 4095. Element AO%(1) controls the first analog output. Background software will transfer the contents of AO%() to the proper analog output whenever Basic writes to the AO%() array. The outputs are 4-20 ma. current loop compatible. The current loop drivers can withstand 100 VDC, and will work linearly over 12 to 48 VDC loop voltages. Figures 3.2A and 3.2B present analog output schematics. Figure 3.2C presents example connections. As with analog inputs, analog outputs must be calibrated before accurate use can be expected. The code in section 3.3 will do this for you.

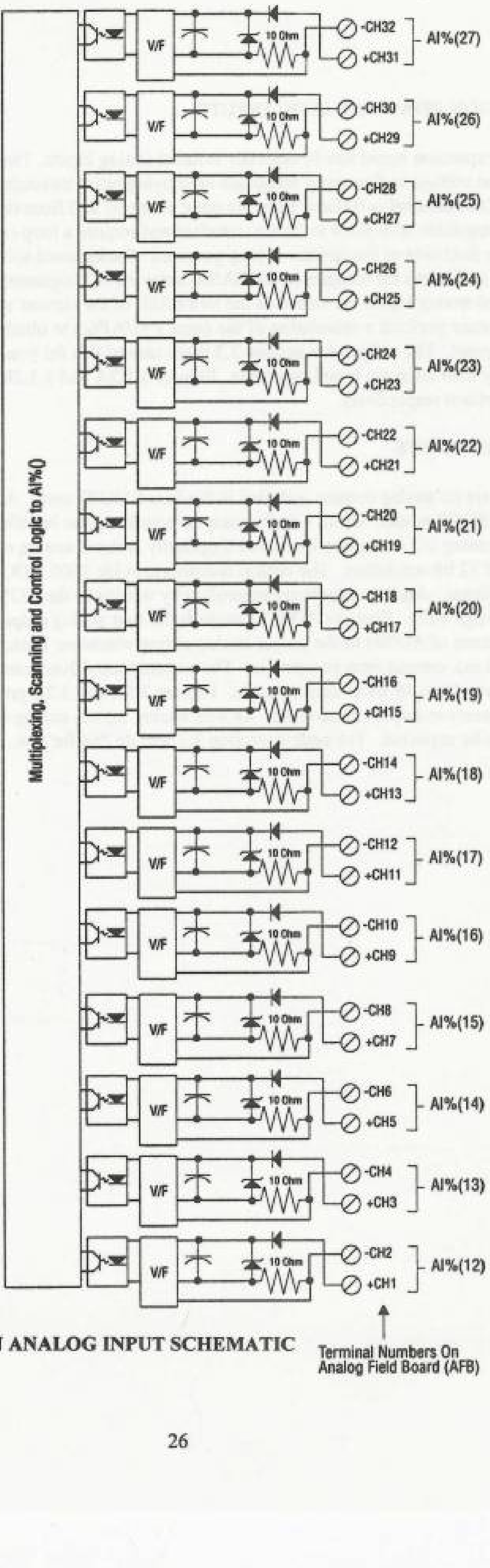


FIGURE 3.1.2A EXPANSION ANALOG INPUT SCHEMATIC

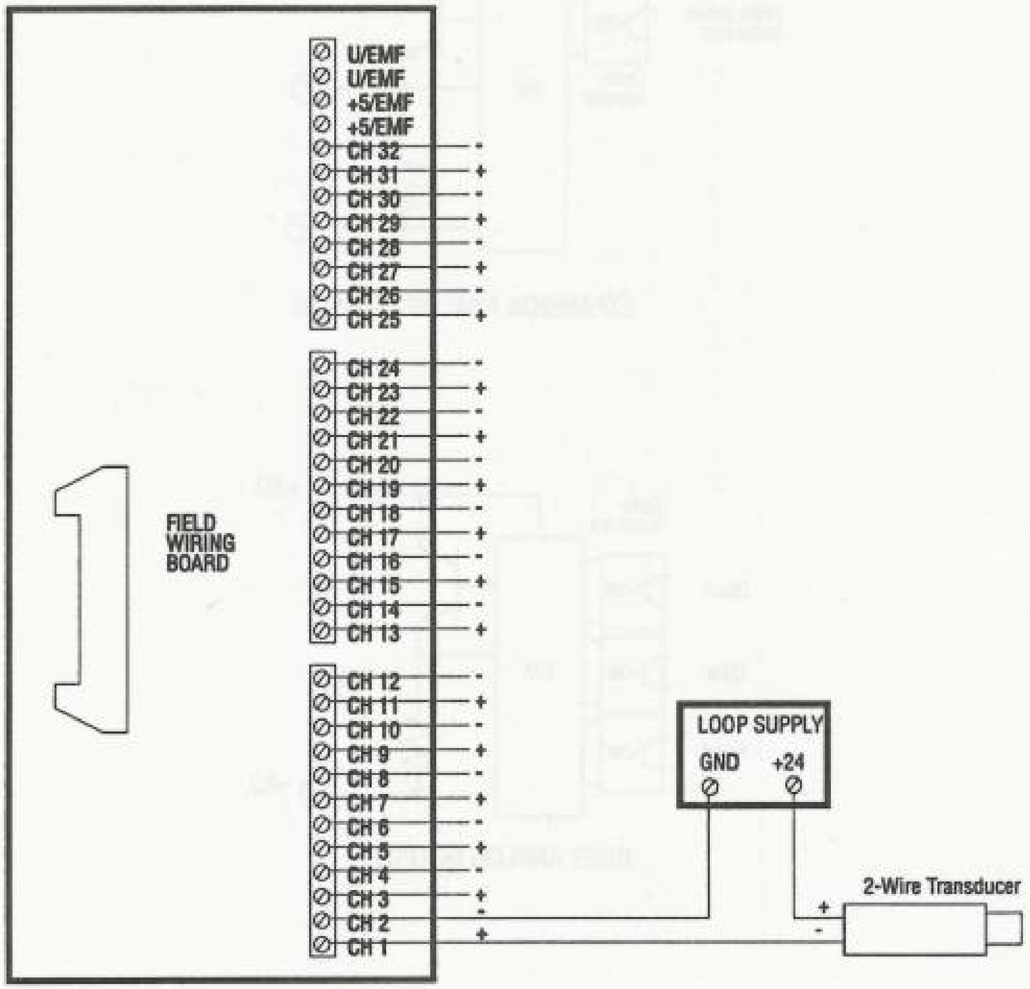
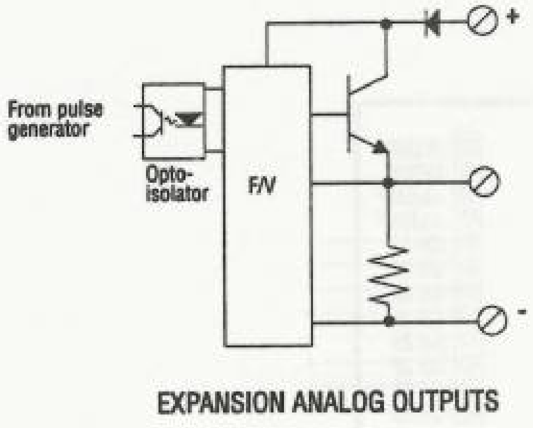


FIGURE 3.1.2B EXPANSION ANALOG INPUT CONNECTIONS



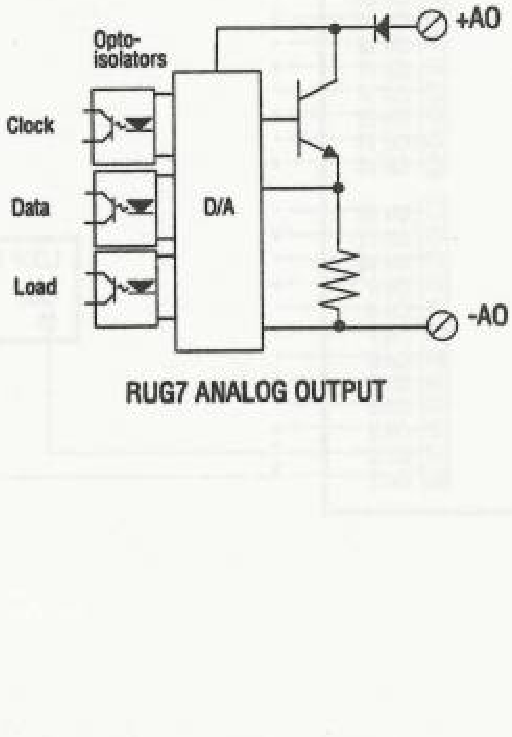


FIGURE 3.2A EXPANSION ANALOG OUTPUT SCHEMATIC

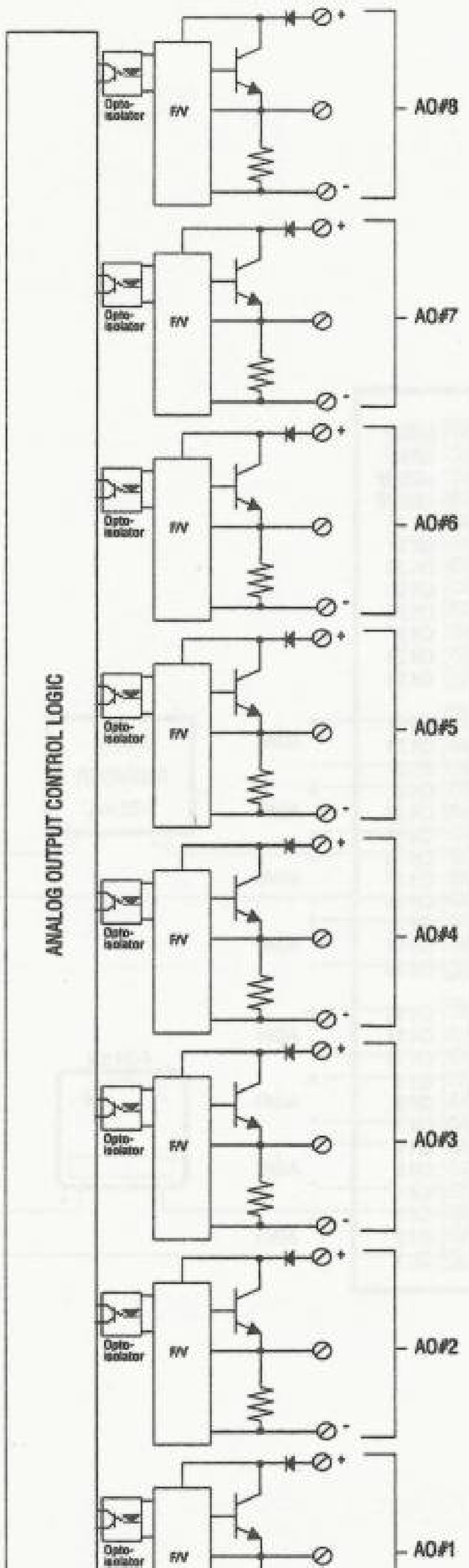


Figure 3.2B EXPANSION ANALOG OUTPUT SCHEMATIC (RUG6 & 8)

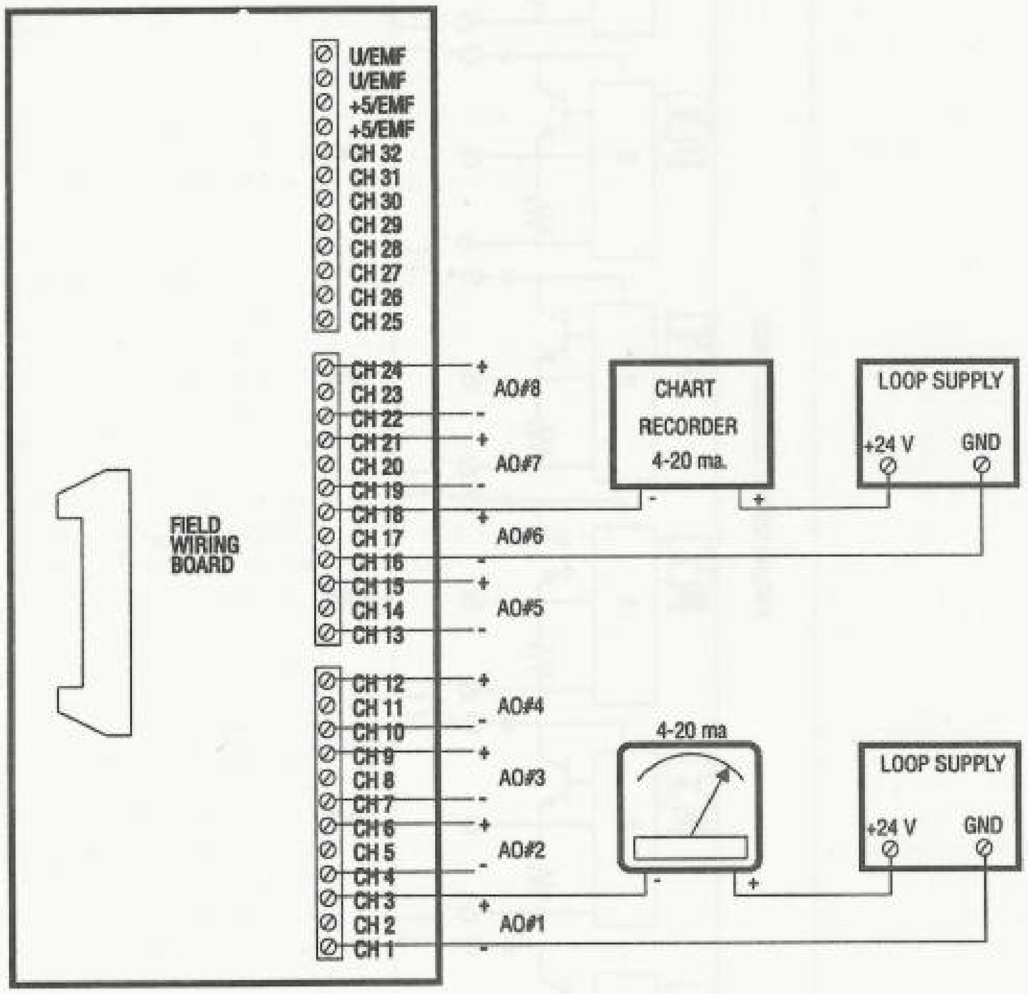


FIGURE 3.2C EXPANSION ANALOG OUT EXAMPLE CONNECTIONS

3.3 CALIBRATION SOFTWARE

Since analog inputs from the A/D converter consist of raw counts corresponding to the analog input values present at any time, it is necessary to convert the raw counts to meaningful engineering units values such as feet, PSI, GPM, etc. In order to do this, the unit must be told the correspondence between the analog input raw counts and the desired engineering units representation during a calibration phase; and then must apply the resulting span and offset factors during realtime operation. Therefore, software necessary for complete analog input handling consists of three routines. In these routines, AY() contains the engineering units result of the calculation; AM() and AB() are the span and offset respectively established during calibration; and AI%() is the raw count from the A/D converter.

CALIN...establishes the values of AM() and AB() for analog inputs; and sets the values of OM() and OB() for analog outputs. This should be done whenever the unit is changed or a sensor is changed or adjusted in order to re-establish the correspondence between the raw counts and the desired resulting engineering units values.

CALINIT... initializes AM() and AB() for use in the equation AY(I)=AM(I)*AI%(I) +AB(Y) by reading their values from the user protected RAM space and writing them to AM() and AB(). For analog outputs, CALINIT reads user protected RAM and saves the values into OM() and OB(). This routine should be exercised once each time the program is booted. The reason for keeping these values in the protected area is so that when you execute a CLEAR to erase all variables, the calibration data will not be effected. Otherwise, every time you executed a CLEAR, you would have to calibrate all the analog inputs.

RTCAL...calculates the engineering units values (feet, PSI, etc.) from span and offset constants (AM() and AB() respectively) setup by CALIN during calibration. This should be executed on each program scan to keep the results updated for display, alarm comparison, reporting, etc. Similarly, for analog outputs, RTCAL applies the equation AO%(I)=X*OM(I)+OB(I) to convert an engineering units value in X to the raw count needed by the analog output driver.

```
RTCAL if SP(2)<1 then SP(2)=1                           'Analog sampling
      X=1/SP(2)                                         'SP(2)=Filter time constant
      for I=1 to 24                                     '24 analog inputs...
      Y=AI%(I)                                          'Get analog input raw count
      if Y=0 and I>11 then Y=K3                         'Head off div by zero for expAI's
      if I<12 then AY(I)=(AY(I)+X*(Y*AM(I)+AB(I)))/(1+X)  'LPF base AI's
      if I>11 then AY(I)=(AY(I)+X*(AM(I)/Y+AB(I)))/(1+X)  'LPF expan AI's
      if AY(I)>K3 then AY(I)=K3                         'Watch high range...K3=32767
      if AY(I)<0 then AY(I)=0                           'Watch low range
      Y=AY(I)*HU                                        '*100 for TLM...HU=100
      if Y>K3 then Y=K3                                 'Range limit
      AR%(1,I+9)=Y                                      'Save to TLM
      next
      for I=1 to 8                                      'Now for 8 analog outputs...
      X=OM(I)*AR%(1,I+6)/HU+OB(I)                       'Get tank level & make raw count
      if X<1 then X=1                                   'Watch range for AO's
      if X>K4 then X=K4                                 'Limit max to 4095
      AO%(I)=X:next                                     'Send to AO%
      return
```

```
'CALMAN calibration routines
'Interacts with user to gather coefficients for 2 point calibration

'For analog input calibration:
'        M and B are stored in AM() and AB() respectively
'        To avoid losing cal when editing program, the calibration
'        constants are stored in the reserved area from
'        $1F00 through $1FFF.  These are loaded by CALINIT upon
'        boot up.  A total of 32 analog channels can be stored in this
'        area.  This routine allocates space for 24 analog inputs, and
'        8 analog outputs.  They are arranged thus:
'                  $1F00-$1F03=AM(1)*65536
'                  $1F04-$1F07=AB(1)*65536
'                  $1F08-$1F0B=AM(2)*65536
'                        .
'                        .
'                  $1FC0-$1FC3=OM(1)*65536
'                  $1FC4-$1FC7=OB(1)*65536

'In realtime operation, result would be:
'        result(I)=AM(I)*AI%(I)+AB(I) for inputs 1 to 11
'        result(I)=AM(I)/AI%(I)+AB(I) for inputs 12 and up.

'For analog output calibration:
'        M and B are stored in OM() and OB() respectively

'In realtime operation, result would be:
'        AO%(I)=OM(I)*value(I)+OB(I)

'CALINIT reads reserved user area and gathers saved cal coefficients.
'        Should be executed at beginning of program.

CALINIT for I=1 to 11
        II=(I-1)*8:gosub CALRAMINX                     'Get AM() cal value to X
        AM(I)=X/65536                                  'Scale down and save
        II=(I-1)*8+4:gosub CALRAMINX                   'Get AB() cal value to X
        AB(I)=X/65536                                  'Scale down & save
        next
        for I=1 to 4
        II=(I+23)*8:gosub CALRAMINX                    'Get OM() cal value to X
        OM(I)=X/65536                                  'Scale down & save
        II=(I+23)*8+4:gosub CALRAMINX                  'Get OB() cal value to X
        OB(I)=X/65536
        next
        return

CALRAMINX J=7936+II                                    'Get X from reserved area, byte J
        X=peek(J)+256*peek(J+1)+65536*peek(J+2)
        XX=peek(J+3) and 128                           'Get sign
        II=peek(J+3) and 127                           'Get MS byte, no sign
        X=X+16777216*(II)
        if XX=0 then return
        X=-X:return
```

```
CALRAMOTX J=7936+II:I1%=0                               'Save X to location in reserved area, byte J
        if X<0 then I1%=128:X=-X                                                 'Get sign of X
        X=X*65536:XX=int(X/16777216)
        X1%=XX and 127 or I1%:poke J+3,X1%                                        'MS byte (1)
        X=X-XX*16777216:X1%=int(X/65536):poke J+2,X1%                            'Byte (2)*****
        X=X-X1%*65536:X1%=int(X/256):poke J+1,X1%                                'Byte (3)*****
        X=X-X1%*256:poke J,int(X):return                                         'LS byte (4)


CALMAN gosub CLRSCREENC
        print "Calibration routine..."
        print "Enter 1 for analog input."
        print "Enter 0 for analog output."
MOR1 get A$
        if len(A$)=0 goto MOR1
        if A$="1" goto CALIN
        if A$="0" goto CALOUT
        goto CALDONE                                                             'return if not 0 or 1

CALIN gosub CLRSCREENC
        print "Enter input channel to calibrate."
        print "Allowed range is 1 through 11, 0 exits":input X
        if X<1 or X>11 goto CALDONE
        CH=int(X)                                                   'Save channel we're calibrating
        print "Apply known low value to analog input";CH
        print "Key in corresponding engrg. units value.":input X
        A1=X:A2=AI%(CH)                                             'Save it and A/D output
        print "Apply known high value to analog input";CH
        print "Key in corresponding engrg. units value.":input X
        A3=X:A4=AI%(CH)                                             'Save it and A/D output
        if CH>11 and A2<1 goto CALINERR                             'Watch for zero input
        if CH>11 then A2=1/A2:A4=1/A4                               '1/AI%(I) if expansion A/D
        if abs(A2-A4)>.000001 goto COMPUT
CALINERR print "Input value error...restarting cal."
        for I=1 to 1000:next                                        'Delay
        goto CALIN
COMPUT M=(A1-A3)/(A2-A4)                                            'Compute M
        B=A1-M*A2                                                   'and B

'Now show result of this calibration
        gosub CLRSCREENC                                            'Clear the LCD
        A2=100:A3=0.5                                               'Use for significance control
CALOOP CR%=0:CC%=0                                                  'Set display cursor location
        print "Present AI";CH;"=";AI%(CH);"  "                      'Show user A/D input
        if CH<12 then A1=int((M*AI%(CH)+B)*A2+A3)/A2                'Control significance
        if CH>11 then A1=int((M/AI%(CH)+B)*A2+A3)/A2                'Expansion AI's
        print "Engrg. value=";A1;"  "                               'Show resulting value
        print "Key in 1 to save"
        print "Key in 2 to discard"
CAL1 get A$
        if len(A$)=0 goto CALOOP                                    'Watch for keystroke
        if A$<>"1" goto CALIN
        AM(CH)=M:AB(CH)=B                                           'Save coefficients
        X=AM(CH):II=(CH-1)*8:gosub CALRAMOTX                        'Put in reserved area
        X=AB(CH):II=(CH-1)*8+4:gosub CALRAMOTX          'Ditto
        goto CALIN                                      'Return for more

CALDONE print "Exiting calibration routines."
        for I=1 to 1000:next:return                     'Delay & exit

CALOUT gosub CLRSCREENC
        print "Enter output channel to calibrate."
        print "Allowed range is 1 through 8, 0 exits.":input X
        if X<1 or X>8 goto CALDONE
        CH=int(X)                                       'Save channel user wants to cal.
        A1=600                                          'Init AO low value
OUTLOOP AO%(CH)=A1                                      'Send value out
        gosub CLRSCREENC
        print "A value of ";A1;"is now present on AO";CH
        print "Key in a new value to get a "
        print "convenient low analog output level."
        print "A negative value continues process.":input X
        if X>=0 then A1=X:goto OUTLOOP
        print "Key in the engrg units value now"
        print "present on analog output";CH:input X
        A2=X
        A3=3800
OUTLOOP1 AO%(CH)=A3                                     'Send value out
        gosub CLRSCREENC
        print "A value of";A3;"is now present on AO";CH
        print "Key in a new value to get a"
        print "convenient high output level."
        print "A negative value continues process.":input X
        if X>=0 then A3=X:goto OUTLOOP1
        print "Key in the engrg units value now"
        print "present on analog output";CH:input X
        A4=X                                            'Get 2nd engrg value
        if abs(A2-A4)>.000001 goto COMPUTO
        print "Input value error...restarting"
        for I=1 to 1000:next                            'Delay
        goto CALOUT
COMPUTO M=(A1-A3)/(A2-A4)                               'Compute M
        B=A1-M*A2                                       'And B

'Now allow user to adjust AO and observe

OUTLOOP2 gosub CLRSCREENC
        A1=int(A3+0.5)                                  'Watch roundoff
        if A1<0 then A1=0                               'Keep AO%>=0
        AO%(CH)=A1                                      'Send to analog output
        print "Present AO%";CH;"=";A1
        print "Present engrg units value is";A4
        print "Key in new EU value for verification."
        print "Negative value exits verification mode.":input X
        A4=X:A3=M*X+B
        if X>=0 goto OUTLOOP2
        print "Key in 1 to save"
        print "Key in 2 to discard calibration.":input X
```

```
if X<>1 goto CALOUT
OM(CH)=M:OB(CH)=B
X=OM(CH):II=(CH+23)*8:gosub CALRAMOTX              'Save to reserved area
X=OB(CH):II=(CH+23)*8+4:gosub CALRAMOTX            'Ditto
goto CALOUT
```

3.4 DIGITAL INPUTS

The RUG6 protected field board has 16 dry contact or logic compatible digital inputs. RUG7 and RUG8 models have 8 channels each. Each digital I/O expansion board has 32 digital inputs. Background software samples each digital input whenever BASIC accesses the DI%() array. All digital inputs are mapped into DI%() beginning with DI%(1) and continuing until the array dimension is exceeded. A contact closure or low logic level on the digital input will result in a 1 in DI%(); an open contact or logic high value will result in a 0 value in DI%(). Figure 3.4A presents the RUG6 digital input schematic. Consult Figures 3.0A, 3.0B, and 3.0C for digital inputs examples for RUG6, RUG7, and RUG8, respectively. Figure 3.4 B presents example connections for expansion digital inputs. Note that all units have one LED per channel and 100K ohm isolation resistors. The resistors protect the inputs from voltage reversal and excessive voltage.

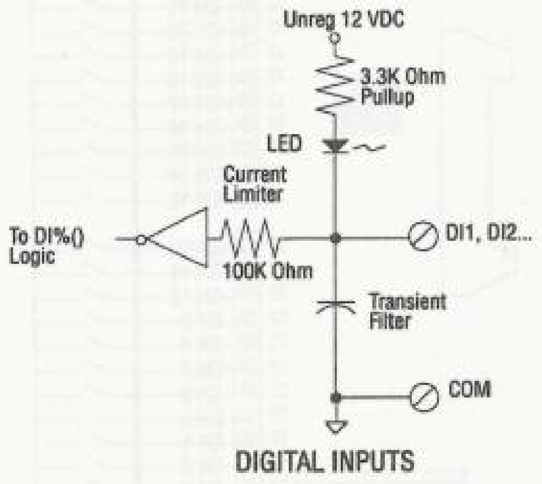


FIGURE 3.4A DIGITAL INPUT SCHEMATIC

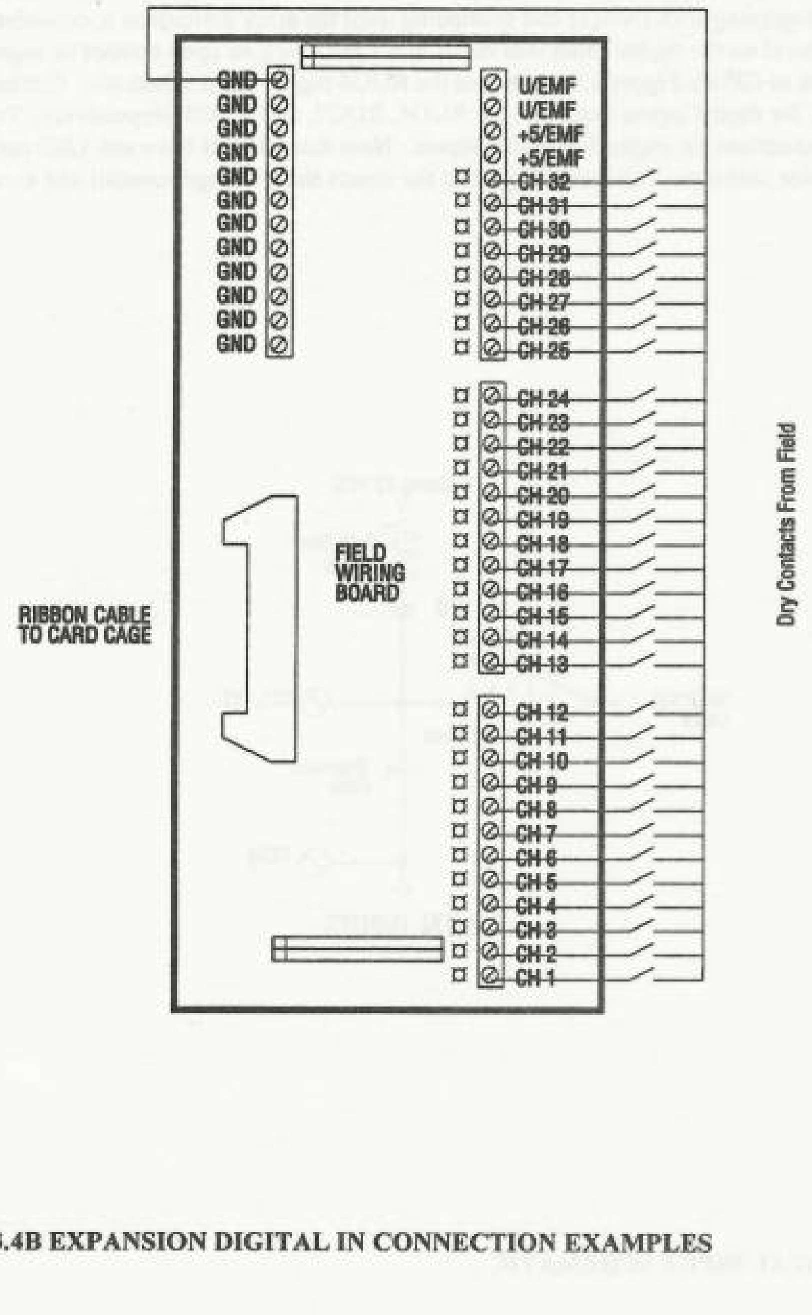


FIGURE 3.4B EXPANSION DIGITAL IN CONNECTION EXAMPLES

3.4.1 DIGITAL INPUT PULSE COUNTING

Background software will count high to low transitions of the digital inputs and store the counts in array DX%(). The inputs are sampled 225 times per second. The number of digital inputs sampled is established by the dimension of the DX%() array, but is limited to 64 total. The counts can be read by BASIC by reading the DX%() array. The counts can be preset by writing to the DX%() array. If the count exceeds 32,768 the value will become negative, but counting will continue. This capability is useful for measuring values from pulse type flow meters and for trapping momentary button pushes by operators. To detect a momentary button push or switch closure, instead of watching the digital input, have the Basic program watch for a nonzero value in the DX%() entry corresponding to the digital input connected to the switch. When the DX%() value is nonzero, the button has been pushed. Write a zero into the DX%() entry to reset it for the next switch closure.

3.4.2 PULSE DURATION I/O

3.4.2.1 PULSE DURATION INPUTS

In addition to counting pulses occurring on the digital inputs, RUGID can convert pulse duration inputs present on digital inputs 1 through 8 to analog values proportional to the time the digital input is turned on. Basically, RUGID background software looks to see if the array PD%() is dimensioned. If so, the first 8 digital inputs are sampled 56 times per second. When a digital input turns on, a counter is started and allowed to run until the digital input turns off. At that time, the counter value is transferred to the PD%() array, indicating how many 56ths of a second the input was on. All that is necessary to initiate this mode is to dimension the PD%() array.

3.4.2.2 PULSE DURATION OUTPUTS

Digital outputs 1 through 8 and 17 through 48 can be made to generate repetitive variable duty cycle pulses by writing to the DU%() array. Array element DU%(0) sets the repetition rate in 56ths of a second; and elements DU%(1) through DU%(48) set the period, in 56ths of a second, within the repetition interval that the corresponding digital output is energized. Any digital outputs for which the DU%() entry is zero are uneffacted by this operation. Those outputs for which DU%() has a nonzero entry will turn on at the beginning of a cycle. Each will turn off as its individual timer times out based upon the value in its DU%() entry. The following example program segment illustrates the operation:

```
.
.
DU%(0)=15*56            'Set overall cycle interval to 15 sec.
DU%(3)=3.9*56           'Output 3 on for 3.9 sec.
DU%(4)=12.3*56          'Output 4 on for 12.3 sec.
.
.
```

In this example, every 15 seconds digital outputs 3 and 4 will turn on simultaneously. After 3.9 seconds, output 3 will turn off. After 12.3 seconds, output 4 will turn off. The range of the DU%() entries is 1/56 second (17.86 ms) to 32767/56 seconds (9.75 minutes).

If DU%(0)=0, the digital outputs act as individual one-shots, not triggered by an overall cycle. In that case, any DU%() entry set non-zero will turn on immediately and turn off after its individual timer has timed out, independently of the other outputs.

3.4.3 OPTICAL ENCODER INTERFACE (ACRO Transducers)

The first 8 digital inputs can be set up to interface with up to four ACRO encoders by dimensioning the AC% array. A pair of digital inputs is required for each encoder. The encoders must be set up to give position pulses on one output and direction indication on the other. They must be connected to the RUGID digital inputs with the position pulses coming in on the odd digital inputs (DI%(1), DI%(3)...), and the direction indications coming in on the even digital inputs (DI%(2), DI%(4)...). For example, to connect a single ACRO transducer to the RUGID, connect its pulse output to digital input DI%(1) and its direction output to digital input DI%(2). The position will then be indicated as a count in array entry AC%(1) and will increase with clockwise rotation. Entries in AC%() indicate the number of transitions sensed from the encoder pulse output, not the number of pulses. These are integer numbers that must be multiplied by the distance represented by each pulse transition, and offset by the initial count in AC%() to obtain engineering units values. Be sure that the slew rate of the transducer shaft is not so fast that pulses narrower than 10 ms result or they could be missed. Note that since the encoders generally put out a 0 to 5 volt logic signal, in order to avoid harming the transducer, the LED should be clipped out on any digital input channels connected to the transducers, and to avoid capacitive loading, each channel's surge protection capacitor should also be removed.

3.4.4 RUG8 PULSE COUNTING DURING SLEEP MODE

The RUG8 has a processor that remains running during SLEEP mode to count time and input pulses. When the RUG8 is awake, the array CO%() is used to set and read the counters as defined in the following table:

Table 3.4.4 PIC COUNTER CO%() ENTRIES

CO%() ENTRY	FUNCTION
CO%(0)	DI #8 up counter, LS
CO%(1)	DI #7 up counter, LS
CO%(2)	DI #6 up counter, LS
CO%(3)	DI #5 up counter, LS
CO%(4)	DI #4 (tipper bucket) up counter, LS
CO%(5)	AI11 (anemometer) up counter, LS
CO%(6)	
CO%(7)	
CO%(8)	DI #8 up counter, MS
CO%(9)	DI #7 up counter, MS
CO%(10)	DI #6 up counter, MS
CO%(11)	DI #5 up counter, MS
CO%(12)	DI #4 (tipper bucket) up counter, MS
CO%(13)	AI11 (anemometer) up counter, MS
CO%(14)	
CO%(15)	
CO%(16)	DI #8 down counter
CO%(17)	DI #7 down counter
CO%(18)	DI #6 down counter
CO%(19)	DI #5 down counter
CO%(20)	DI #4 (tipper bucket) down counter
CO%(21)	AI11 (anemometer) down counter
CO%(22)	
CO%(23)	Sleep timer, one count per second
CO%(24)	DI #8 encoder #2 down counter
CO%(25)	DI #7 encoder #2 up counter
CO%(26)	DI #6 encoder #1 down counter
CO%(27)	DI #5 encoder #1 up counter
CO%(28)	
CO%(29)	
CO%(30)	
CO%(31)	

3.5 BASE DIGITAL OUTPUTS

The base digital output complement consists of relay driver digital outputs capable of driving up to 100 ma. each when all are on, or up to 500 ma. when only one is on. They are compatible with up to 48 volt relays. Most channels are equipped with a 10 amp relay per channel with an LED per channel. Digital outputs can be controlled by writing to array DO%(). Array element DO%(1) controls the first digital output. Writing the following values cause the following actions:

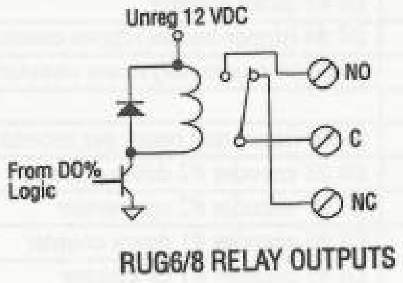
DO%()=	ACTION
0	Relay off
1	Relay on
2	Relay flashes on/off every 1/2 second

The following table defines the functions of the relay drive channels for the different RUGID models:

Table 3.5 RUG6, RUG7, RUG8 RELAY CHANNELS

DO%() CHANNEL	RUG6	RUG7	RUG8
DO%(1)	RELAY #1	RELAY #1	RELAY #1
DO%(2)	RELAY #2	RELAY #2	RELAY #2
DO%(3)	RELAY #3	RELAY #3	RELAY #3
DO%(4)	RELAY #4	RELAY #4	RELAY #4
DO%(5)	RELAY #5		ON=puts unit to sleep
DO%(6)	RELAY #6		
DO%(7)	RELAY #7	ON=Enable hi rate battery charge	
DO%(8)	RELAY #8	ON=turn on display backlight	

Figure 3.5A presents a digital output relay drive schematic.



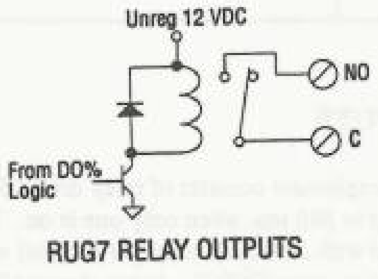


FIGURE 3.5A MAIN BOARD DIGITAL OUTPUT SCHEMATIC

3.5.1 EXPANSION DIGITAL OUTPUTS

For RUG6's and RUG8's, the digital output complement can be expanded by adding one or more digital output expansion boards inside the card cage, and attaching a digital field wiring board (DFB)to each. Each digital output expansion board/digital field wiring board combination adds 32 digital output relay drivers. The drivers are capable of driving up to 48 volt DC loads at up to 100 ma. each. The first digital output on the wiring board, labeled CH1, can be controlled by writing to DO%(17), CH2 can be controlled by writing to DO%(18), etc. Note that the drivers are open collector Darlington drivers that sink current to ground when energized. Figure 3.5.1 illustrates the proper connections for relays, lamps and LED indicators.

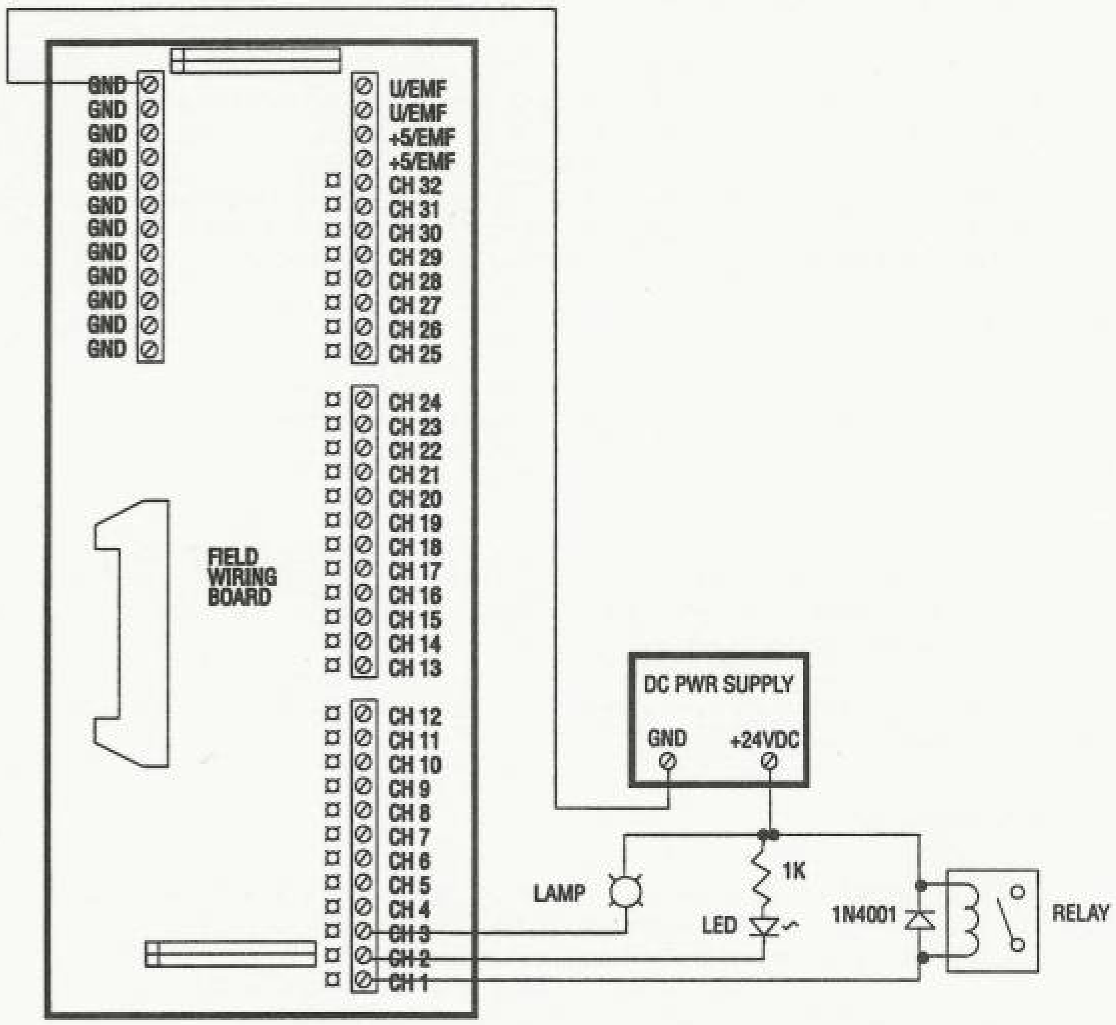


FIGURE 3.5.1 EXPANSION DIGITAL OUTPUT CONNECTIONS

3.5.2 EXPANSION RELAY OUTPUTS

For greater drive capability than the relay drivers discussed above, RUG6's and RUG8's can employ relay field boards (RFB's) instead of the digital field boards (DFB's). Each RFB has eight 10 amp relays and its own relay drivers and LED's. Up to four RFB's can be daisy chained together and connected to one MIO or DIO in the card cage. Therefore, one unit can be expanded by as much as 128 relays above the base relay complement.

3.5.3 PULSE DURATION OUTPUTS

Digital outputs 1 through 8 and 17 through 48 can be made to generate repetitive variable duty cycle pulses by writing to the DU%() array. Array element DU%(0) sets the repetition rate in 56ths of a second; and elements DU%(1) through DU%(48) set the period, in 56ths of a second, within the repetition interval that the corresponding digital output is energized. Any digital outputs for which the DU%() entry is zero are uneffacted by this operation. Those outputs for which DU%() has a nonzero entry will turn on at the beginning of a cycle. Each will turn off as its individual timer times out based upon the value in its DU%() entry. The following example program segment illustrates the operation:

```
        .
        .
DU%(0)=15*56                  'Set overall cycle interval to 15 sec.
DU%(3)=3.9*56                 'Output 3 on for 3.9 sec.
DU%(4)=12.3*56                'Output 4 on for 12.3 sec.
        .
        .
```

In this example, every 15 seconds digital outputs 3 and 4 will turn on simultaneously. After 3.9 seconds, output 3 will turn off. After 12.3 seconds, output 4 will turn off. The range of the DU%() entries is 1/56 second (17.86 ms) to 32767/56 seconds (9.75 minutes).

If DU%(0)=0, the digital outputs act as individual one-shots, not triggered by an overall cycle. In that case, any DU%() entry set non- zero will turn on immediately and turn off after its individual timer has timed out, independently of the other outputs.

3.6 LCD DISPLAY

The RUG6 LCD display is a 64 dot by 256 dot display capable of displaying 8 lines by 42 characters, or presenting plotted data. Upon power application, the display will be in the graphic mode. In this mode, all graphic and character modes described below are supported. Characters may be interspersed with plotted data by writing character data after the plot is done. In EPROM versions 3.2V and later, a character mode is added that does not support graphics but presents character data to the display more than nine times faster than the graphic mode, and enables access to the kana and Greek character set built in to the LCD driver. The display mode is controlled by the MSB of location $0225 (549), which should be poked by BASIC and followed by a form feed character (chr$(12)). The following code segments can be used to set the two modes:

To set graphics mode:
```
CLRSCREENG poke 549,0:PO%=0:print chr$(12);:CC%=0:CR%=8:return
```

To set character mode:
```
CLRSCREENC poke 549,128:PO%=0:print chr$(12);:CC%=0:CR%=0:return
```

The modes can be set at any time. Generally, its best to leave the display in character mode unless you wish to trend to the LCD. For more detail on the character mode see paragraph 3.6.7.

3.6.1 CONTROLLING CONTRAST

Display contrast may vary with temperature so it may need adjusting from time to time. You can adjust it by holding the "8" key and the decimal point (".") keys down simultaneously; or holding the "9" key and the decimal point down dimultaneously. It is also software controllable by writing to the contrast register at location $0216 (534). The range of values permissible is 0 through $1F (31). There is approximately 1 second delay between the adjustment by keyboard or by controlling the contrast register, so you should adjust the contrast in small increments and wait for the contrast to settle. The contrast register value will be retained through a power outage.

3.6.2 WRITING TO DISPLAY

You can write to the display by setting PO%=0 and then executing the PRINT statement. For example, to print "Hello" on the LCD, execute the following:

```
100 PO%=0:PRINT "Hello"
```

3.6.3 CLEARING THE SCREEN

You can clear the screen by writing a form feed to the LCD as follows. Its best to use the CLRSCREENG or CLRSCREENC subroutines above since they also establish the display as being in graphic or character mode also, and return the cursor to the top of the screen.

```
100 PO%=0:PRINT CHR$(12)
```

3.6.4 CONTROLLING THE CURSOR IN GRAPHIC MODE

Characters directed to the display will be written to the dot locations specified by writing to variables CR% (cursor row) and CC% (cursor column). The cursor position specifies where the lower left corner of the next character will be written. Each character written will cause the cursor location to increment by 6 dots (one character width). A line feed will cause the cursor to move down 8 dots; and a carriage return will move the cursor to the left side of the current line. The upper left corner of the display is location CR%=0 and CC%=0. The cursor position cannot be read by reading variables CC% and CR%. Instead, a peek to the following locations must be done:

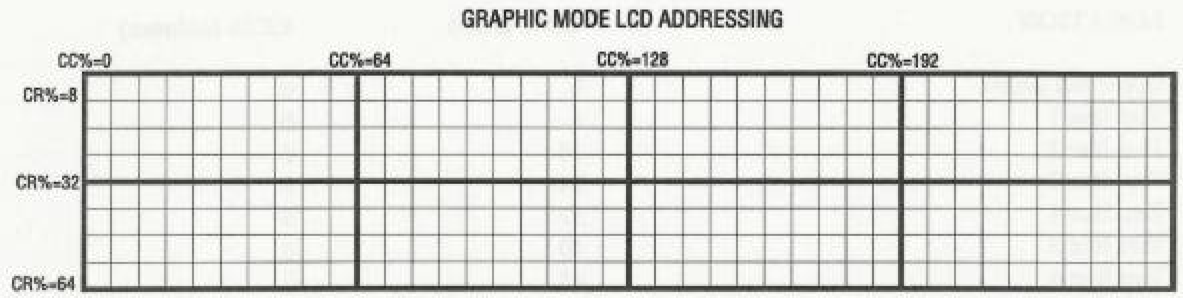
```
Column position=peek(1235)
Row position=peek(1236)
```

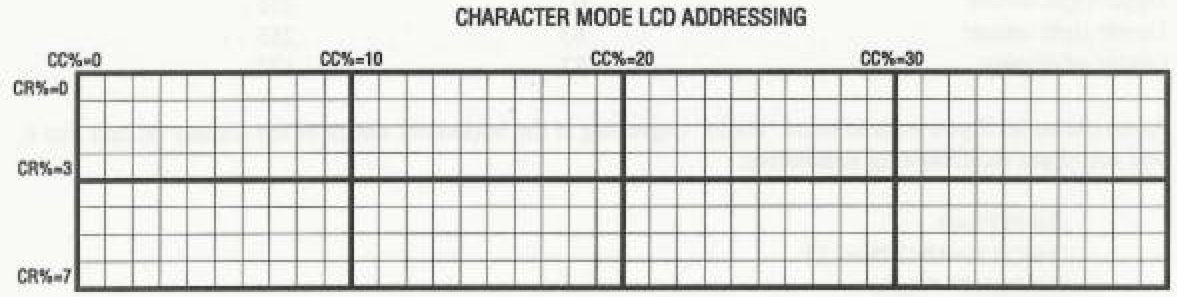
The following table gives useful locations on the display:

LOCATION	**CR% (row)**	**CC% (column)**
Upper left corner	0	0
Text line 1	8	0
Text line 2	16	0
Text line 3	24	0
Text line 4	32	0
Text line 5	40	0
Text line 6	48	0
Text line 7	56	0
Text line 8	64	0
Lower left corner	64	0
Upper right corner	0	255
Lower right corner	63	255
Center of display	32	127

As an example, if you wish to write "Hello" beginning at the horizontal center of the display on text line 6, you could use the following statements:

```
100 PO%=0
110 CR%=48:CC%=127
120 PRINT "Hello"
```





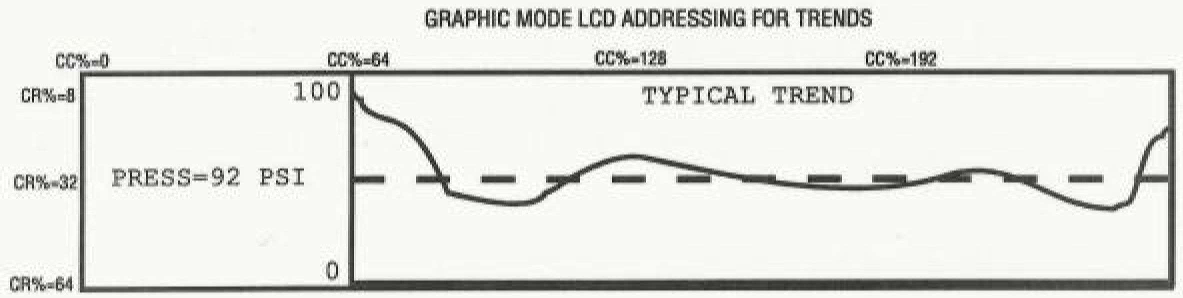


FIGURE 3.4 LCD CURSOR POSITIONING EXAMPLES

3.6.5 PLOTTING DATA (GRAPHIC MODE ONLY)

The PLOT function can be used to plot data horizontally on the LCD. Plots with 256 bit horizontal resolution and 64 bit vertical resolution can be produced. Generally this will be used for amplitude versus time plots that simulate strip chart recorders, with amplitude on the vertical scale, and time on the horizontal scale. Fixed point arrays P1%(I), P2%(I), and P3%(I) contain the data that will be plotted when the PLOT function is invoked. The location of the lower left hand corner of the plot axes is specified by the cursor control variables CC% and CR% as indicated above. The plot will occupy the portion of the display above and to the right of CC% and CR%, extending to the right until the end of the display is reached or the end of the P1% array is reached. Therefore, the entire display will be occupied by the plot if the lower left hand corner is specified for the origin (CR%=63:CC%=0), and P1% is dimensioned as DIM P1%(256). By specifying a value greater than zero for CC%, space will be reserved to the left of the plot for scale values, etc. It is generally recommended that space not be reserved below the plot since that will diminish the vertical resolution of 64 dots.

The values contained in the arrays P1%, P2% and P3% must be in the range of vertical resolution of the plot to avoid incorrect plot results. If the plot is specified with CR%=63, then values contained in the arrays must be in the range of 0 to 63. The leftmost value plotted will be that contained in location 1 of the arrays, e.g., P1%(1). P1% will always be plotted with a solid line. P2% and P3% can be plotted solid or hatched by specifying an 8 bit mask in locations P2%(0) and P3%(0) respectively. Note that if P2%(0)=0, then P2% will be blank; if P2%(0)=255 then P2% will be shown as a solid line. The system will present a vertical line with alternate dots on if a value of 128 is added to the normal value of an entry. This is useful for showing a time tick. The following example will use P1% to plot a cosine wave, P2% to show the 0 value of the cosine wave, and P3% to indicate the sign of the cosine wave. A time tick is shown on dot 31. An 8 character wide area is reserved to the left of the plot for text.

```
DIM P1%(256), P2%(256), P3%(256)
P2%(0)=127                                '7 bits on, one off hatch
P3%(0)=255                                'Sign plot solid
FOR I=1 to 256
P1%(I)=30+25*COS(I/10)                    'The cosine save
P2%(I)=30                                 'The 0 value of wave
IF P1%(I)=>P2%(I) THEN P3%(I)=5           'Wave above 0
IF P1%(I)<P2%(I) THEN P3%(I)=2            'Wave below zero
NEXT
P1%(31)=P1%(31)+128                       'Time tick
CC%=48                                    'Reserve 6X8 dots for text
CR%=63                                    'Full height plot
B=PLOT(1)                                 'Call plot function
```

3.6.6 PLOTTING BARGRAPHS (GRAPHIC MODE ONLY)

Bargraphs can be plotted to the LCD by using the builtin graphic characters CHR$(138) through CHR$(143). The following code segment (BARGRAPH.LIB) will produce a horizontal bar 8 pixels high with the bar length equal to X. Figure 3.6.6 below presents the graphic character set implemented in the RUGID's graphic display mode.

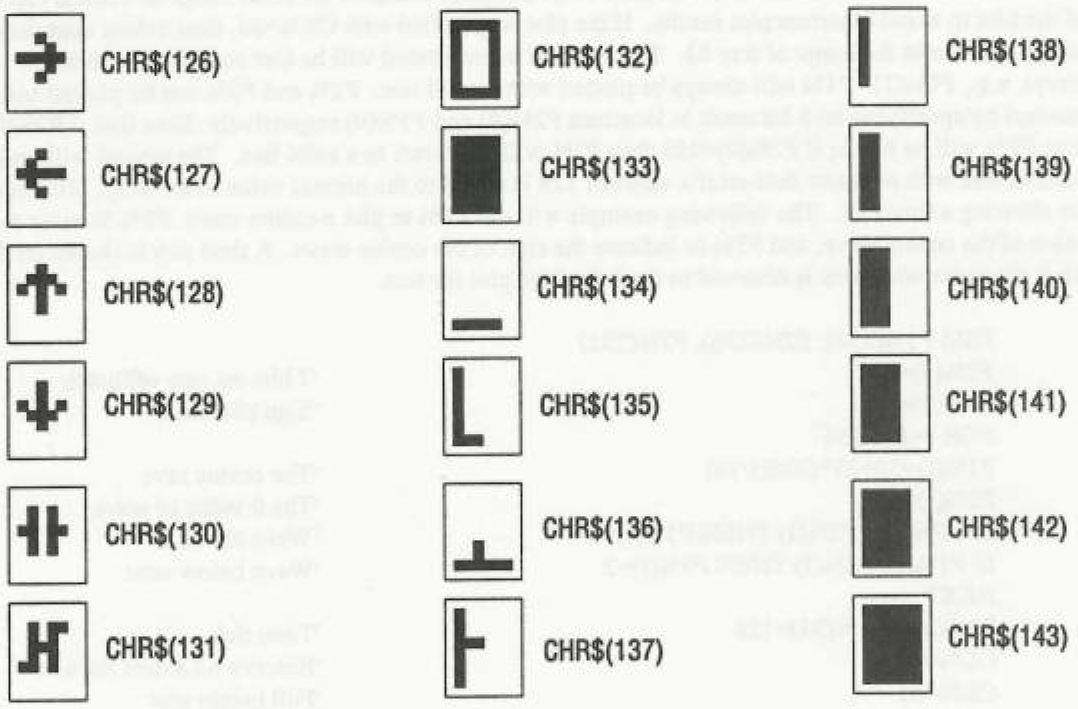


Figure 3.6.6 GRAPHIC CHARACTER SET

```
'BARGRAPH.LIB...routine to plot X as bargraph
'Starts plotting from present cursor position
'Does NOT range check X.
'If bar is to start other than at the left side of the display,
'then the '250' in BAR2 should be diminished by the number of
'pixels the start of the bar is offset from the left of the LCD.

BARGRAPH PO%=0:Y=int(X/6)                   'Plot X as hor. bar starting from cursor
         YY=int(X-Y*6)                      'Y=solid blocks, YY=pixels extra
         if Y<1 goto BAR1                   'Watch for no solid blocks
         for I=1 to Y:print chr$(143);:next 'Solid part of bargraph
BAR1 if YY=0 goto BAR2                      'Any partial?
         print chr$(137+YY);                'Show partially filled character
BAR2 X=int((250-Y*6)/6)                     '# chars to blank after bar
         if X<1 then return
         for I=1 to X:print " ";:next:return 'Blank rest of bar
```

3.6.7 LCD CHARACTER MODE

On EPROM versions 3.2V and later, the system supports a character mode that provides faster character writing to the display and enables access to the display's special kana and Greek characters. This mode does not support trend plotting and bargraph displays. It can be enabled by executing the following subroutine:

To set character mode:

```
CLRSCREENC poke 549,128:PO%=0:print chr$(12);:CC%=0:CR%=0:return
```

This routine will install the character mode, clear the LCD, and set the cursor to the upper left corner of the display. Note that cursor control is different for this mode. Since only characters can be written, the range of cursor control variables is CC%=0 to 40, CR%=0 to 7. The cursor position cannot be read by reading variables CC% and CR%. Instead a peek to the following locations must be done:

```
Column position=peek(1235)
Row position=peek(1236)
```

3.6.7.1 ACCESSING SPECIAL CHARACTERS

With the character mode, the LCD's builtin character generator is used to write the screen, enabling access to the kana and Greek characters listed in the table in appendix X. To provide for access to these characters using standard ASCII codes, the RUGID operating system can be made to translate the ASCII codes beginning with the [@] symbol up to the special character area by writing a value into location $0225 (549) that is added to the ASCII code to specify the special character code. (A value of 80 is also added by the operating system to extend the translation range.) The following example will translate the ASCII codes up to the first column of kana codes, enabling the four columns of alpha ASCII codes to access the four columns of kana characters:

```
poke 549,128+16
```

In this example, the [128] maintains the character mode, the [16] tells the operating system to add 16+80 to the alpha character codes before writing to the LCD.

3.6.8 DISPLAY BACKLIGHT CONTROL (RUG7 ONLY)

RUG7 models are equipped with backlit displays. To control the backlighting, execute the following lines:

```
DO%(8)=0            'Turns backlight OFF
DO%(8)=1            'Turns backlight ON
or DO%(8)=2         'Makes backlight flash on and off
```

3.7 KEYBOARD

The 16 key membrane keypad interfaces to the unit using a ribbon cable. The following table presents the ASCII codes generated by the keyboard and presented to BASIC in normal operation:

KEY	ASCII CODE
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
"."	"."

"-"	"-"
Up arrow	U
Down arrow	D
ENTER	CR
CLEAR	C

In addition, certain key combinations have special meaning:

KEY COMBINATION	MEANING
"." + 8	LCD contrast up
"." + 9	LCD contrast down

Figure 3.7.1 presents the standard keyboard arrangement.

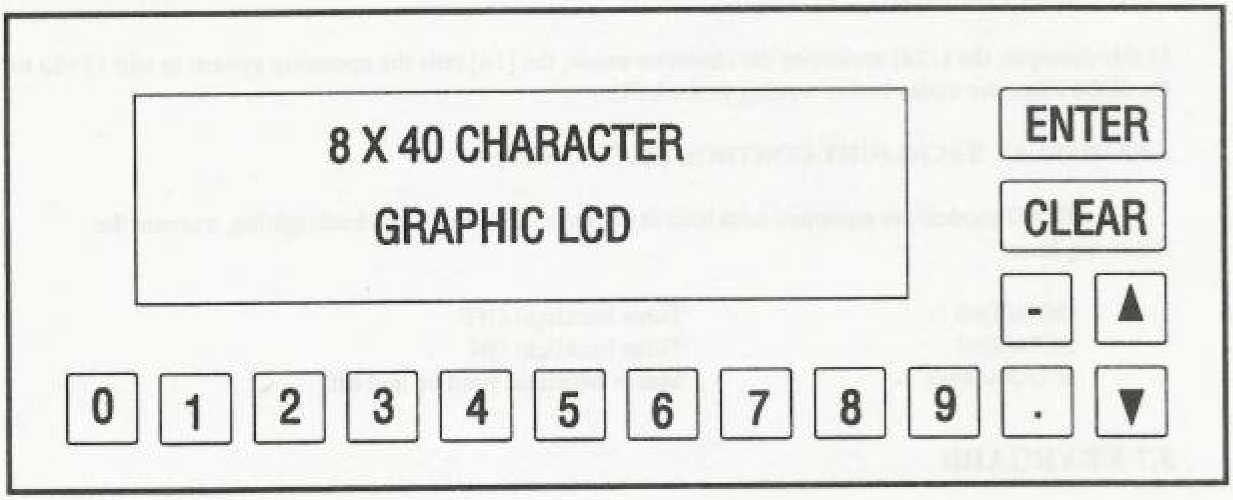


FIGURE 3.7.1 KEYBOARD

3.8 SERIAL PORTS

Each unit has a single serial port in the main board that is configured as RS232. This port is generally used for program loading and accessing the unit from a local terminal or computer. You access this port from BASIC by setting PO%=1 and then using the PRINT command. If you have ordered the modem option, the unit will have a second serial port that you access by setting PO%=2 and then using the PRINT command. This port can be configured as RS232, 2-wire modem or 4-wire modem as defined in section 3.8.3.

3.8.1 COMMUNICATIONS OPTIONS SETTING

Serial port data rates and other options can be set from the monitor mode by hitting the Z key and then responding to the prompts as necessary to establish the desired communications operating parameters. These options and others can also be set under software control. The easiest way to set modem port parameters is to use the DIAGNOSX.CMP program, delivered with the unit. The table below defines the communications options present in the unit as delivered:

Parameter	Terminal Port	Modem port
Baud rate	9600	300
Word length	8 bits	8 bits
# stop bits	1	1
Parity	Disabled	Disabled
Modem function	---	Answer
Interface	RS232	Bell 103

Be careful of altering the data rate using a terminal, as that would make communications with the terminal impossible unless the terminal data rate is altered to match. All communications parameters are kept in voted nonvolatile memory and loaded into the UARTs upon power application and other times as necessary to support communications. The following sequences illustrate the process of setting up the terminal port to operate at 2400 baud with even parity, an 8 bit data word, and one stop bit.

1) Hit Z key in monitor mode.

2) RUGID computer displays the existing setup:

```
MODEM: 300 BAUD; 1 STOP BIT; ANSWER
8 BIT WORD; ODD PARITY...CHANGE(Y/N)?(N)
```

(The N says we do not wish to change the modem setup.)

```
TERMINAL: 9600 BAUD; 1 STOP BIT
8 BIT WORD; ODD PARITY...CHANGE(Y/N)?(Y)
```

(The Y says we wish to change the terminal setup.)

3) RUGID computer displays:

```
ENTER A NEW BAUD RATE: (2400)
```

(The '2400' selects 2400 baud.)

4) RUGID then displays:

```
SELECT # STOP BITS: 1 OR 2 (1)
```

5) Next RUGID displays:

```
SELECT WORD LENGTH: 5, 6, 7, OR 8 BITS (8)
```

(The 8 selects an 8 bit word length.)

6) Finally, RUGID displays:

```
SELECT PARITY: 1=DISABLED, 2=ODD
3=EVEN, 4=MARK, 5=SPACE (3)
```

(The '3' indicates we want even parity.)

7) RUGID will display "DONE" when finished.

The new communications parameters will take effect immediately upon entry of the parity selection. Had we been configuring the modem port, additional inquiries to establish the modem's mode of operation would have been made as follows:

```
SELECT MODEM FUNCTION: A=ANSWER
B=ORIGINATE, C=HI TONES, D=LOW TONES
E=AUTOSET (E)
```

(The A response sets the modem for automatic tone setting depending upon whether the modem answered or originated an autodial/autoanswer call.)

Again for the modem only, RUGID will prompt:

```
SELECT # RINGS FOR AUTOANSWER
USE 0 FOR NO AUTOANSWER (3)
```

The 3 will cause RUGID background software to answer on the third ring.

You should use the ANSWER mode if the RUGID computer is to be called by another computer or modem; the ORIGINATE mode if the RUGID computer will be calling another computer; and HI TONES or LOW TONES if you wish several computers (more than two) to communicate among each other bidirectionally. For radio use, or for applications where RUGID's are to talk to one another, you should use LOW TONES to minimize the effect of companders, and to enable any unit to talk to any other. Note that since HI TONES and LOW TONES options use only one frequency pair or the other for communications, only half duplex communications will be possible. Also, the AUTOSET mode only applies to communications in the monitor mode.

Each port has separate 80 character receive and transmit ring buffers that are accessed by the interrupt process as well as the BASIC interpreter. When writing the transmit port, BASIC fills the buffer and the interrupt process empties it at the UART compatible rate. When reading the port, the interrupt process writes the buffer from the UART at the UART compatible rate, and BASIC can then read it. This minimizes the possibility that data will be lost at high data rates. Variable PO% controls which port is currently open according to the following table:

PO%	PORT IN USE
0	LCD display and keyboard
1	Terminal port
2	Modem port
3	Printer
4	Speech synthesizer
5	Touch tone detector/dialer

3.8.2 COMMUNICATIONS WATCHDOG

If a character has not been received on a serial port within the last 255 seconds, a software watchdog will re-initialize the UART associated with that port. Therefore, programs that expect to receive data infrequently on port 2 should retrigger this watchdog frequently by executing the following statement:

```
100 POKE 1146,255
```

3.8.3 SELECTING PORT TYPE ON MODEM BOARD

You can control the connection of the modem board's UART, modem and speech synthesizer by sending appropriate control characters out port 5. The characters sent select the connections as specified below:

SELECTION	UART/RUG8 TONE SELECT	MODEM	SPEECH
A	Modem/Bell	2-wire	4-wire
B	RS422/ALERT	2-wire	4-wire
C	RS232	2-wire	4-wire
D	None	2-wire	4-wire
E	Modem/Bell	4-wire	2-wire
F	RS422/ALERT	4-wire	2-wire
G	RS232	4-wire	2-wire
H	None	4-wire	2-wire

Some useful commands follow:

```
100 PO%=5                    'Select port 5
110 PRINT "A"                'Standard dialup modem connection
                             'Speech sent to radio transmitter

120 PRINT "E"                'Modem connected to radio,
                             'Speech connected to dialup system

130 PRINT "G"                'Local RS232 serial connection,
                             'Speech connected to dialup system

140 PRINT "F"                'UART to local RS422 network
                             'Speech connected to dialup system
```

3.8.4 TRANSMITTER KEYING CONTROL

You can control the transmitter keying circuit on the modem board by PRINTing to port 5 as follows:

```
100 PO%=5
110 PRINT "Z"          'Transmitter off
110 PRINT "T"          'Transmitter on
```

If you are using the CRC-secured communications (see section 5) over the 4-wire port, the transmitter is controlled automatically by background software and should not be controlled directly by BASIC using the POKE's above.

3.8.5 TOUCHTONE DIALING

To control the touchtone generator you must first have the modem connected to the 2-wire port (e.g., PO%=5:PRINT "A") because the modem chip has the touchtone generator. Then use the PRINT command to send the digits to be dialed to the touchtone generator. The following program segment will cause the number 555-1234 to be dialed:

```
100 PO%=5                    'Select touchtone port for PRINT
110 PRINT "A"                'Modem connected to 2-wire channel
120 PRINT "5551234"          'Send the digits
```

3.8.6 DETECTING DIAL TONE, BUSY, TOUCHTONES, RINGING

Dialtone and busy signals are detected by the modem IC, whereas touchtones are detected by a dedicated touchtone digital filter IC that is continuously monitoring the 2-wire interface. Therefore, you can detect touchtones at any time, but in order to detect dial tone and busy signals, the modem IC must be connected to the 2-wire interface. To read the status of the tone detectors, set PO%=5 and execute a GET A$. The character returned indicates the state of tone reception as indicated below:

CHARACTER RETURNED	MEANING
"0"-"9" "*", "#"	Received touchtone characters
"B"	Busy signal present
"C"	Connected to distant modem
"D"	Dial tone present
"L"	Low signal at modem
"N"	No tones present
"R"	Incoming ring signals detected
"X"	Ringback detected

Sometimes you may want to more directly test for carrier presence at the modem. You can do this by executing a X=(PEEK(964) and 16). If X is zero, then no carrier is present. Up to 20 received touchtone characters will be accumulated and buffered by background software, so before looking for touchtone reception, be sure to flush the buffer by executing GET(A$) in succession.

RUGID's background software will do the work necessary to detect ringing signals. If at least one ring has been detected in the preceding 10 seconds, then the statement PO%=5:GET A$ will return a "R" character indicating that ringing is present. The number of rings detected is counted in location $03CE (974). If 10 seconds elapse without a ring, the ring counter will be cleared and the ring indication "R" will no longer be returned on port 5. Note that if RUGID is hooked to a 2 wire dialup channel, and RUGID is not to answer the phone, then your Basic program must keep the count detector register cleared or else background software may answer the phone. You can accomplish this by executing the following statement periodically:

```
100 POKE 974,0
```

3.8.7 CONTROLLING ONHOOK/OFFHOOK

Normally, when waiting for an incoming phone call, the phone will be "onhook", i.e., the dialing relay is open. Before dialing a call, the phone must be "offhook", i.e., the dialing relay must close. This triggers the phone system to issue a dial tone. Once the dial tone is present, dialing may begin. To control onhook/offhook, set PO%=5 and use PRINT "K" for onhook (idle) and PRINT "L" for offhook (dialing, etc.).

3.8.8 PORT 5 COMMAND/CHARACTER SUMMARY

The following table summarizes the single character command and tone detection assignments used by port 5 to control the modem port/speech synthesizer interface.

COMMAND CHARACTER	FUNCTION
"A"	UART to modem 2-wire (RUG8 Bell tones); speech 4-wire
"B"	UART to RS422 (RUG8 ALERT tones); speech 4-wire
"C"	UART to RS232; speech 4-wire
"D"	UART disconnected; speech 4-wire
"E"	UART to modem 4-wire(RUG8 Bell tones); speech 2-wire
"F"	UART to RS422 (RUG8 ALERT Tones); speech 2-wire
"G"	UART to RS232; speech 2-wire
"H"	UART disconnected; speech 2-wire
"K"	2-wire phone interface onhook
"L"	2-wire phone interface offhook
"W"	Send modem answer tone
"X"	Modem transmitter on
"Y"	Modem transmitter off
"T"	Transmitter key on
"Z"	Transmitter key off

RECEIVED CHARACTER	MEANING
"0"-"9"	Touchtone received characters
"*", "#"	
"B"	Busy signal present
"C"	Connected to distant modem
"D"	Dial tone detected
"L"	Low signal level
"N"	No tones present
"R"	Ringing signals detected
"X"	Ringback present

3.8.9 MODEM SETUP OVERRIDE

The modem serial port, port #2, setup parameters installed in voted RAM will be used to establish the baud rate, word length, etc., upon power application, and any time the unit is reset by the watch dog timer. You can override the setup for the modem port from BASIC by using the MODE command followed by parameters that specify the new baud rate, word length, etc. The correct syntax for the MODE command is:

MODE=BAUD,PARITY,WORDLENGTH,STOPBITS,ANS/ORIGINATE

The options for each parameter are as follows:

PARAMETER	OPTIONS
Baud rate	50,75,110,134,150,300,600,1200,2400 3600,4800,7200,9600,19200
Parity	E (even), O (odd), N (none), M (mark), S (space)
Word length	5,6,7,8
Stop bits	1,2
Ans/originate	A (answer), O (originate), L (low tones)

As an example, the following program segment will set the communications parameters as indicated:

```
PO%=5:print "MODE=2400,E,8,1,A"
```

Resulting setup would be: 2400 baud, even parity, 8 bit word, 1 stop bit, answer mode. These parameters will take precedence over the previous setup immediately. If any parameter is detected to be in error, the setup in place prior to issuance of the MODE statement will remain in place.

3.8.A RUG8 PORT 2 RS232 PORT

RUG8 units have an optional RS232 port on their COM boards to enable serial communications with external modems, particularly special purpose modems to enable satellite access. The RS232 port is accessible from the underside of the unit using a 10 pin shrouded header on a ribbon cable with a DB9 connector on the other end to connect to the external modem. The connector pinout is listed below. The cable is available from RUGID. Note that pin 1, when asserted true, will wake up the RUG8. This can be used to wake up the unit and enable communications in applications where the units are generally asleep to conserve power.

Table 3.8.A RUG8 Optional RS232 Port Pinout

PIN	FUNCTION
1	DCD Data Carrier Detect
2	RXD Received Data (incoming)
3	TXD Trensmitted Data (outgoing)
4	DTR Data Terminal Ready
5	GND Ground
6	NC
7	RTS Request To Send
8	CTS Clear To Send
9	NC
10	GND Ground

3.9 PRINTER PORT (RUG6 & 8)

The printer port is a standard parallel Centronics compatible type port that is accessed by setting variable PO% to 3 before executing the PRINT statement. It can also be enabled for generating program listings by hitting the "P" key while holding down the CONTROL key. Hitting "Q" while holding down the CONTROL key will disable the printer for program listing purposes. It does not effect the use of the printer for BASIC PRINT instructions. The printer port is on the RUG6 protected field board and on the RUG8 low power field board. The pinout is presented below and is identical to that used by IBM PC compatible computers; so a standard DB25 male to Centronics connector will enable the RUGID to be connected to standard parallel printers.

Table 3.9 PRINTER PORT PINOUT

PIN	SIGNAL
1	Strobe
2	Data 0
3	Data 1
4	Data 2
5	Data 3
6	Data 4
7	Data 5
8	Data 6
9	Data 7
10	ACK
11	Busy
12	Paper out
13	NC
14	NC
15	Error
16	NC
17	NC
19-25	GND

3.9.1 PRINTER ERROR DETECTION

Attempting to print to a printer that is disconnected, off line, out of paper, or turned off can cause your program to stop while background software waits for an acknowledgement from a printer. To minimize this possibility, RUGID monitors error signals from standard Centronics printers and makes them available to Basic to test before issuing print statements. The easiest way to sample this register is to use the routine PRWATCH in the library file PRWATCH.LIB.

PRINTER STATUS BYTE: location $8760=34656

	X	X	X	X	PE	BUSY	ERR	X
OK					0	0	1	
Cable off					1	1	1	
Out of paper					1	X	X	
Busy of off line					X	1	X	
Out of paper or off line					X	X	0	

3.10 TIMERS

An array of timers, DT%(), is available for use in general purpose event timing. The number of timers is set by the dimension statement. Once per second, the system will decrement any DT%() entry that is not zero. You can program time delays by setting a DT%() variable to the number of seconds you wish to delay, and then testing for that timer reaching zero. When it reaches zero, the time delay has expired. There are 64 timers maximum possible, DT%(1) through DT%(64). The dimension statement should be set for no more timers than necessary in order to avoid having background software process timers unnecessarily. The timers have a range of 0 through 32767 seconds.

3.11 REALTIME CLOCK/CALENDAR

The internal real time clock/calendar can be set in the monitor mode by hitting the "T" key and then entering the date and time in the exact format with which they are displayed. It is important to keep this roughly accurate even if not used to time BASIC data gathering or other functions because it is used to tag the occurrence of the last BASIC execution error.

Monitor software controls the clock interface, sampling the clock once per second and storing its outputs in array CK%(I), and transferring CK%(I) to the clock when BASIC writes to CK%(I). Therefore, when running a BASIC program, you only need to read and write to CK%(I) to interface to the realtime clock/calendar. The following table defines the assignments of CK%(I):

ARRAY ENTRY	MEANING	RANGE
CK%(0)	Seconds 0-59	
CK%(1)	Minutes 0-59	
CK%(2)	Hours	0-23
CK%(3)	Day of Mo.	1-31
CK%(4)	Month	1-12
CK%(5)	Year	0-99
CK%(6)	Day of Wk.	1-7

The clock/calendar accounts for leap year.

3.11.1 SETTING CLOCK/CALENDAR FROM MONITOR

Type "T" (User's entry to invoke time/date display and allow time or date changing.)

RUGID replies:

```
CURRENT DATE/TIME IS SAT 01/11/86 21:45
SET WITH SAME FORMAT:_
```

You may now enter a "RETURN" if you do not wish to alter the time/date setting; or you may enter a new time directly below that shown beginning with the character designated by the underline prompt. Note that this is a 24 hour clock. All essential information must be entered in the format indicated. The acceptable mnemonics for the days of the week are: SUN MON TUE WED THU FRI SAT.

3.12 WATCHDOG TIMER

In order to account for transient errors and software errors, a hardware watchdog timer is provided that if allowed to time out will reset the computer and, therefore, restart whatever software process was executing before the error occurred. The watchdog timer is retriggered by the interrupt process which is triggered by the BASIC program. If interrupts fail to generate a trigger, the timer will reset the processor 0.57 seconds later. A counter is maintained by the interrupt process that stretches the retrigger interval for BASIC to 2.3 seconds. The watchdog timer is retriggered automatically by the monitor when in the monitor or BASIC command mode. When running a BASIC program, a POKE 530,1 must be executed at least once every 2.3 seconds to avoid an inadvertent reset. For debugging purposes, the timer may be disabled by executing a POKE 531,0. Once debugging is complete, however, the timer can be enabled by executing a POKE 531,1. The watchdog timer is a powerful device for assuring continued operation especially in unattended installations. Whether enabled by Basic or not, the watchdog will catch most system errors, including those detected by the Basic interpreter.

3.13 MEMORY WRITE PROTECTION

RAM memory containing the program and other critical data including communications setup parameters is write protected, which means it cannot be modified if the program is running without taking action to first open the memory. We recommend against opening the RAM if at all avoidable, as the program can be corrupted by transients during this time. The procedure for opening and closing memory is presented below:

To open RAM:

```
 A=PEEK(969) AND 191:POKE 969,A:POKE 34800,A
(Wait 3 seconds, then RAM will be open.)
```

To close RAM:

```
 A=PEEK(969) OR 64:POKE 969,A:POKE 34800,A
(RAM is closed immediately.)
```

3.14 MEMORY EXAMINE/MODIFY

In monitor mode, any memory location can be examined by hitting the "M" key and then entering in hexadecimal the address to be examined. The monitor will display eight bytes beginning with that address along with their ASCII equivalents if possible. Any location that is not write protected or inherently unwritable can be altered by then hitting the "/" key and keying in the new contents for any of the eight bytes displayed. Extreme caution should be used since the executive software uses some locations to store flags and data. As many bytes as desired can be entered in sequence, even in excess of the eight displayed. The RETURN key terminates the alteration process. The space key causes the unit to display the next 8 bytes. Holding the space key down continuously causes the display of an additional 8 bytes approximately every second.

3.15 SPEECH SYNTHESIZER

The speech synthesizer is controllable only from a BASIC program. The synthesizer is voice trainable , and in fact must be trained by you before it can perform any useful speech output. Memory on the speech synthesizer board can hold approximately 1 minute of speech. With a fully populated RAM bank board, that capacity can be increased to a total of 4 minutes. Messages are tagged with numbers in the range of 1 to 999 when recorded. Those tag numbers are used to reference the recorded messages during playback or deletion. Messages must be deleted before they can be recorded.

You control the speech synthesizer by setting PO%=4, and then sending the commands listed below using the PRINT statement. Commands can be strung together and will not be executed until a carriage return is sent. Synthesizer status can be read by setting PO%=4 and then executing the GET A$ command. A single ASCII character at a time will be returned.

The following commands are supported:

COMMAND	ACTION
C	Cancel all strings and halt recording/playback.
Dxxx	Delete message xxx. Recover free space.
Pxxx	Playback message xxx.
Rxxx	Record message xxx.
#(-)xxx.x	Speak the value (minus)xxx.x For this to work, messages 1 through 12 must be: Msgs 1..9 = "One...Nine" Message 10 = "Zero" Message 11 = "Point" Message 12 = "Minus"
CR	Execute preceding string of commands.

The following reply strings are supported:

REPLY	MEANING
E#	Error encountered: #1=out of memory. #2=referenced message already recorded #3=referenced message not recorded #4=no message number given
I	Synthesizer idle.
M###	Memory remaining=### Kbytes.
W	Working on command string.

The program segment called TRAINSP.LIB, in the software library included with each unit, can handle all speech training, playback and deletion functions.

3.16 RAM BANK CONTROL

RUGID's RAM bank expansion card can be used to store data in data logging applications, or to store speech information to extend the amount of speech storage from the 60 seconds available on the speech board to more than 240 seconds. The board is available with 32K byte static RAM chips installed as indicated in the following table:

CHIPS	CAPACITY	SPEECH STORED
8	256K	69 sec.
16	512K	138 sec.
20	640K	173 sec.

An onboard lithium battery powers the board during power outages, and will maintain memory contents for up to 18 months of cumulative power outage. Note that one RAM bank board cannot be split between data storage and speech storage, and that only one board can be used for each of those applications. Therefore, two RAM bank boards can be installed in a RUG-6 or RUG8 unit, but one must be used for data storage and the other for speech storage.

When used for speech storage, the RAM bank board will be automatically detected and used by the speech synthesizer. No special attention is necessary by the programmer.

When used for data storage and retrieval by BASIC, the board appears as a pair of bank select registers and a 256 byte RAM window that can be PEEKed and POKEd. The BASIC programmer must control the bank selection by POKEing to the CHIP SELECT and PAGE SELECT registers defined below, and then either write or read the 256 addresses in the resulting RAM window.

An additional capability of the RAM bank board is that it can be write protected, write enabled, or placed under software control using jumper J2 on the board. When under software control, bit 6 of the chip select register controls write protection.

CHIP SELECT REGISTER: location=$8100 (33024)

The following table defines contents of this register for accessing the various chips installed on the board. See section 8.2.1 for RAM bank component locations.

REG. VALUE	CHIP SELECT	REG. VALUE	CHIP SELECT	REG. VALUE	CHIP SELECT
$70=112U1	$68=104	U9	$58=88	U17	
$71=113U2	$69=105	U10	$59=89	U18	
$72=114U3	$6A=106	U11	$5A=90	U19	
$73=115U4	$6B=107	U12	$5B=91	U20	
$74=116U5	$6C=108	U13			
$75=117U6	$6D=109	U14			
$76=118U7	$6E=110	U15			
$77=119U8	$6F=111	U16			

PAGE SELECT REGISTER: location=$8110 (33040), range is 0 to 127.

RAM WINDOW: location=$8000 (32768) to $80FF (33023)

This is the range of addresses in which you actually store the data using POKE 32768+AD,XX where AD is the address (0 through 255) for one byte of your data, and XX is the data (0 through 255). To read the data back, use PEEK (32768+AD).

3.16.1 STRING STORAGE & RETRIEVAL FROM RAM BANK

Strings can be stored and retrieved to/from the RAM bank using BASIC's string operators and pokes per character, but that is slow compared to using the built in string transfer subroutines built into the operating system and described here. Basically, using the built in routines, transfers occur between the reserved string, Z$ and the RAM bank at addresses set in the bank select registers by BASIC. Once the bank selects are setup, BASIC must call an EPROM subroutine using the USR() function to trigger the transfer.

3.16.2 STORING TO RAM BANK

Use the following procedure to store a string to the RAM bank:

1 Transfer the string to Z$.

2 Set the chip select register, e.g., POKE 33024,112.

3 Set the page select register, e.g., POKE 33040,35.

4 Set the byte within the RAM bank's window where you want the first byte of Z$ to be stored:

```
POKE 1420,64.
```

5 Setup the USR() function to point to the string storage routine:

```
POKE 4,27:POKE 5,255
```

6 Execute the USR instruction to send the string Z$ to the RAM bank:

```
X=USR(0)
```

The string Z$ will now be stored in the above example on chip #1, page 35, beginning at location 64. It will be stored with a "0" appended to the end to indicate the end of the string.

3.16.3 RETRIEVING A STRING FROM THE RAM BANK

Retrieving a string is almost identical:

1 Set the chip select register, e.g., POKE 33024,112.

2 Set the page select register, e.g., POKE 33040,35.

3 Set the byte within the RAM bank's window from where you want the first byte of Z$ to be retrieved, e.g.,

```
POKE 1420,64.
```

4 Setup the USR() function to point to the string retrieval routine:

```
POKE 4,30:POKE 5,255
```

5 Execute the USR instruction to read the string from the RAM bank and save it in Z$:

```
X=USR(0)
```

6 Transfer the string in Z$ to another string, because, if you change RAM bank register, the contents of Z$ will then reflect the contents of the new window presented by the RAM bank.

3.17 PLOTTING TO PRINTER

RUGID's operating system has background software to assist in plotting any combination of up to 128 analog and digital data sets to dot matrix printers. A RAM bank board with at least 128K of RAM is required to hold the samples to be plotted and to store the plot image that RUGID produces and spools out to the printer. Basically, what's required is that the program:

1) store samples in any of up to 128 designated areas,
2) specify the size of the plot and location on the sheet,
3) specify any grid lines to be plotted,
4) identify which sets of samples are to be plotted,
5) specify the dot pattern to be used for each trace,
6) issue the command PO%=3:X=plot(1)
7) monitor location 1390 (peek(1390)) to see when done,
8) perform a backward form feed, and print any text legends.

In order for plotting to work properly, you must have a dot matrix printer connected to RUGID's printer port that is capable of performing a backward form feed. Otherwise text legends cannot be written to the plot after the analog data is plotted. The software examples presented below work with Star Micronics' NX1000, NR10 and NR15 printers.

3.17.1 RAM BANK USE FOR PLOTTING

Before issuing the command to plot, parameters must be stored in the RAM bank to setup the plot characteristics such as plot size, location, trace pattern, etc. Note that the RAM bank is segmented into 24 chips, 128 pages per chip, and 256 bytes per page. The first two chips (64 K bytes) are reserved for the plot image and print spooling by RUGID. The first page of the 3rd chip contains setup parameters that you must install before issuing the plot command. Remaining pages are used to store the data samples to be plotted, with 6 pages reserved per plot. Therefore, each plot can consist of up to 678 data points with each point being a two byte number in the range of 0 to 639. The table below specifies locations containing necessary information to enable RUGID to perform the plot. Before you feel mired in the details of this table, note that the code segments supplied in the following sections perform most of the table fill in for you.

TABLE 3.17.1 RAM BANK USE FOR PLOTTING TO PRINTER

CHP	PAGE	256 BYTE WINDOW
112	0-	Reserved for plot image and spooler
113	127	
114	0	Byte 0=Vertical size of plot in bytes (1 to 79) Byte 1=Horizontal size of plot in bytes (1 to 95) Byte 2=Spare Byte 3=Spare Byte 4=Vertical axis location, bytes from upper left corner of sheet (0 to 79) Byte 5=Horizontal axis location, bytes from upper left corner of sheet (0 to 95) Byte 6=Spare Byte 7=Spare
		Byte 8=Pattern of plot #1, 0=don't plot . . 255=solid line, 127=dashes, 85=fine dots, etc. . Byte 135=Pattern of plot #128, 0=don't plot
		Bytes 136-143=locations of solid horizontal grid lines in byte pairs LS, MS Bytes 144-151=locations of dashed horizontal grid lines in byte pairs, LS, MS Bytes 152-175=locations of dotted horizontal grid lines in byte pairs, LS, MS
		Bytes 176-183=locations of solid vertical grid lines in byte pairs, LS, MS Bytes 184-191=locations of dashed vertical grid lines in byte pairs, LS, MS Bytes 192-255=locations of dottd vertical grid lines in byte pairs, LS, MS
114	1 2 3 4 5 6	0,1...254,255=plot #1 samples 0 to 127 LS,MS 0,1...254,255=plot #1 samples 128 to 255 LS,MS 0,1...254,255=plot #1 samples 256 to 383 LS,MS 0,1...254,255=plot #1 samples 384 to 511 LS,MS 0,1...254,255=plot #1 samples 512 to 639 LS,MS 0,1...254,255=plot #1 samples 640 to 767 LS,MS
114	7 8 9 10 11 12	0,1...254,255=plot #2 samples 0 to 127 LS,MS 0,1...254,255=plot #2 samples 128 to 255 LS,MS 0,1...254,255=plot #2 samples 256 to 383 LS,MS 0,1...254,255=plot #2 samples 384 to 511 LS,MS 0,1...254,255=plot #2 samples 512 to 639 LS,MS 0,1...254,255=plot #2 samples 640 to 767 LS,MS
114	13- 18	Plot #3 samples...

Additional RAM is used to store samples for up to 128 total plots...

3.17.2 SETTING PLOT SIZE AND LOCATION

The plotting software in EPROM sets the printer for a plot density of 60 dots/inch horizontally, and 72 dots per inch vertically. Each plot can be 80 bytes high (640 dots) by 768 dots wide. Therefore, the maximum plot size is 8.89 inches high by 12.8 inches wide. This will only fit on 14 inch wide printer paper. If you're using an 8.5 inch printer, you will be limited to 510 horizontal data points maximum. If you wish to leave space to the left of your plot for vertical scale information, you must reduce the horizontal size of your plot correspondingly to keep from running off the right side of the paper. Similarly, if you wish to leave space at the top or bottom of the plot for legends, you must reduce the vertical size of the plot. The table below will assist in setting the 4 bytes that establish the size and location of the plot for RUGID's background software.

TABLE 3.17.2 EXAMPLES SPECIFYING PLOT SIZE AND LOCATION

CHP	PAG	BYTE	PLOT RESULT
114	0	0=79 1=63 4=79 5=0	Full size plot, 8.5 inch paper
		0=79 1=95 4=79 5=0	Full size plot, 14 inch paper
		0=79 1=87 4=79 5=7	Full vertical size, 8 bytes reserved to right of plot, 14 inch paper
		0=79 1=87 4=79 5=0	Full vertical size, 8 bytes reserved to right of plot, 14 inch paper
		0=73 1=87 4=73 5=7	8 bytes reserved to left and below plot, 14 inch paper

The following code segment installs the four bytes necessary to setup plot size and location for a full height plot on 14 inch paper with 6 bytes reserved to the left of the plot for legending:

```
.
.
.
CH=114:PG=0:gosub CHIPPAGE                  'Setup chip and page select
AD=0:X=79:gosub RAMIT                               'Set vertical size
AD=1:X=89:gosub RAMIT                               'Set horizontal size
AD=4:X=79:gosub RAMIT                               'Set axis 80 bytes down
AD=5:X=6:gosub RAMIT                                'Set axis 6 bytes from left
.
.
.
CHIPPAGE poke 33024,CH:poke 33040,PG:return         'Chip & page select
RAMIT poke 32768+AD,X:return                        'Write a byte
```

3.17.3 ENABLING GRID LINES AND AXES

RUGID can be made to plot horizontal and vertical axes and grid lines in solid, dashed and dotted form by writing the locations of the lines in the proper locations in chip 114, page 0 as indicated in table 3.17.1 above. Each grid mark location is specified by writing a byte pair (LS,MS) that contains the dot location of the line in relation to the origin (0,0) at the plot's lower left corner. Any byte pair that contains (255,255) will not be plotted. Note from Table 3.17.1 that the following numbers of grid lines are supported:

TYPE	HORIZONTAL	VERTICAL
Solid	4	4
Dashed	4	4
Dotted	12	32

As an example, the following code segment installs grid lines as indicated in the comments:

```
.
.
gosub CLRGRID                                    'Clear grid definitions
AD=136:X=0:gosub RAM2                            'Solid hor. axis at zero
AD=176:X=0:gosub RAM2                            'Solid vert. axis at zero
AD=144:X=SP(5):gosub RAM2                        'Dashed hor. line at SP(5)
AD=146:X=SP(4):gosub RAM2                        'Dashed hor. line at SP(4)
AD=152:X=600:gosub RAM2                          'Dotted hor. line at 600
AD=154:X=400:gosub RAM2                          'Dotted hor. line at 400
AD=192:X=230:gosub RAM2                          'Dotted vert.line at 230
AD=194:X=460:gosub RAM2                          'Dotted vert.line at 460
AD=196:X=690:gosub RAM2                          'Dotted vert.line at 690
.
.
RAM2 XX=int(X/256):X=X-256*XX                    'Split X into 2 bytes
        poke 32768+AD,X:poke 32769+AD,XX         'Store 2 bytes to RAM bank
        return
CLRGRID CH=114:PG=0:gosub CHIPPAGE               'Setup chip & page selects
        X=255                                    'Value to turn off grid line
        for AD=136 to 255:gosub RAMIT:next       'Write X to grid area
        return
.
.
```

3.17.4 ENABLING PLOTS AND SETTING PATTERNS

One byte per plot is used in chip 114, page 0 to tell RUGID which of 128 possible stored plots are to be sent to the printer, and what dot pattern is to be used. As shown in Table 3.17.1, locations 8 through 135 are used for this purpose. Plot #1 is enabled by writing to location 8; plot 2 is enabled by writing to location 9, etc. Writing a nonzero value turns on the plot, with the value specifying the dot pattern. For example, a value of 255 specifies a solid line (all dots on); a value of 127 gives a dashed line (7 bits on, 1 bit off); a value of 85 would turn on every other dot, and a value of zero would turn off that plot. Note that the actual data samples do not have to be moved to use them in plots; simply enabling the plotting by writing to this 128 byte area enables you to mix plots on the paper. For example, the following code segment enables plots 1,2, and 3 as solid lines, and plots 20 and 21 as dashed lines:

```
.
.
CH=114:PG=0:gosub CHIPPAGE                     'Setup chip and page selects
gosub CLRPATTRN                                'Clear pattern specifiers
AD=8:X=255:gosub RAMIT                         'Plot 1 solid line
AD=9:X=255:gosub RAMIT                         'Plot 2 solid line
AD=10:X=255:gosub RAMIT                        'Plot 3 solid line
AD=27:X=127:gosub RAMIT                        'Plot 20 dashed line
AD=28:X=127:gosub RAMIT                        'Plot 21 dashed line
.
.
CLRPATTRN CH=114:PG=0:gosub CHIPPAGE           'Setup chip & page selects
        X=0                                    'Value to turn off pattern
        for AD=8 to 135:gosub RAMIT:next       'Write value to pattern area
        return
.
.
```

3.17.5 STORING DATA FOR PLOTTING

Data samples to be plotted must be scaled to fit on the plot and then stored into the proper one of 128 plot areas. Each plot area consists of 6 contiguous pages of 256 bytes each, with each data sample consisting of two bytes in LS,MS order. When plotting commences, the left-most data point to be plotted will use the value stored in the first byte pair of the 6 page area being plotted. The values that can be plotted on a full height (80 byte) plot can span the range of 0 through 639. If your plot height is less than 80 bytes, the maximum value is less as specified in the following relationship:

Max value=8*(vert bytes)-1

The following example stores a sin wave in plot area number 2. The sin is scaled to occupy the center half of a full vertical size plot.

```
.
.
SINPLOT I=2                                    'Designate which plot
        IX%(I)=0                               'Zero sample index
        for II=1 to 750                        'Save 750 points
        X=320+160*sin(II/20)                   'Compute a sample & scale
        gosub STOREXI                          'Store the sample X in plot I
        next
        return
.
.
STOREXI PG=6*I-5:XX=int(IX%(I)/128)            'Begin page calculation
        PG=PG+XX
        AD=2*(IX%(I)-128*XX)                   'Address in page
        XX=int(PG/128)
        CH=114+XX:PG=PG-XX:gosub CHIPPAGE      'Setup chip & page addresses
        gosub RAM2                             'Store sample to RAM bank
        IX%(I)=IX%(I)+1                        'Increment sample index
        if IX%(I)>767 then IX%(I)=0            'Limit sample index range
        return
.
```

3.17.6 COMMANDING PLOT OUTPUT TO PRINTER

After setting up the plot specification area on the RAM bank and storing the plot data samples, as described in the paragraphs above, you cause the plot generation to commence by executing the following statement:

```
            PO%=3:X=plot(1)                    'Start plotting
```

The PO%=3 tells the plotting EPROM software to plot to the printer as opposed to the LCD (PO%=0); X can be any variable; and the argument of plot() is just a place holder. Upon executing the above statement, the processor will clear chips 112 and 113 on the RAM bank, read the plot setup information, and plot your designated data to chips 112 and 113. It will then send the first 70 bytes of the plot out to the printer transmit buffer and return to your basic program. As your basic program continues to run, after each program line the printer buffer will be examined and, if empty, another 70 bytes will be sent out until the plot data has all been transmitted to the printer. Your program can monitor the plot's completion status by executing:

```
            X=peek(1390)
```

The values returned indicate the following:

X=0	Idle
X=1	Busy with plot
X=2	Error in plot setup

After the plot is finished (peek(1390)=0), you may issue a reverse form feed to send the printer back to the top of the form you just plotted.

```
            print chr$(27);chr$(12);           'Reverse form feed
```

Then print all titles, legends, and any other text you wish on the plot. The following program segment will accurately print legends on the left scale. It does this by changing the printer's form feed to one dot and then stepping down one dot at a time and printing legends at the proper lines with one dot resolution.

```
.
.
LEFTSCALE gosub VERT1DOT                           'Setup for 1 dot form feed
        print:print:print                          'Down 3 dots for alignment
        for AD=632 to 0 step -1
        print
        if AD=600 then print " 15 ft."             'Print '15 ft. at dot 600
        if AD=int(SP(5)) then print "    High alarm=";SP(5);"ft."
        if AD=0 then print "  0 ft."        '0 feet at origin
        next
        gosub VERT16                               'Back to 6 LPI spacing
        return
.
.
VERT1DOT print chr$(27);"A";chr$(1);               'Setup 1 dot form feed
        return
VERT16 print chr$(27);"2";:return                  'Setup 6 LPI form feed
.
```

SECTION 4

4.0 BASIC INTERPRETER

"It is practically impossible to teach good programming to students that have had a prior exposure to BASIC: As potential programmers, they are mentally mutilated beyond the hope of regeneration."

Edsger Dijkstra, 1975

All RUGID units contain our BASIC interpreter in EPROM. We've been asked why we chose BASIC instead of C, PASCAL, FORTH or any of several other languages. Since our charter is to provide low cost units that are easy to configure for field use, we chose BASIC because it's a language that is easy to learn, easy to apply, supports field modification, enables support at reasonable cost by a large body of programmers, and has low RAM occupancy. More than 80% of all code written is in BASIC, indicating that there are many programmers who know some dialect of the language.

Since our BASIC interpreter is supplied in EPROM it does not need to be loaded from an external source in order to run. Only the application program, which contains the instructions that our BASIC interpreter "intreprets", needs to be loaded to put the unit to work. Our BASIC interpreter conforms with standard 8K extended BASIC in all respects except that a PLOT extension has been added to make plotting to the LCD and printer convenient; and certain variables and arrays have been assigned to specific I/O functions so that background software in EPROM can perform scanning.

Short software examples are given throughout this manual to illustrate specific I/O access and statement syntax. However, this manual is not a sufficient tutorial for an inexperienced programmer. Therefore, we recommend that if you are not familiar with BASIC programming, you obtain one of the excellent tutorials on the BASIC language.

4.1 DIRECT AND INDIRECT COMMANDS

The BASIC interpreter supports both direct and indirect commands. Direct commands are executed by keying in a valid statement without a preceding line number. Statements such as

```
PRINT A,B,C
D3=17.5
```

are direct commands. The computer will execute the direct command immediately without effecting the stored program. If a program had been running and had stored values in memory, the stored values would be available to the direct commands. The main utility of direct command execution is to test execution of particular statements and to allow examination of stored values. After executing a direct command the RUGID will prompt you with:

```
Free PGM=12456  Free DATA=4567
OK
```

This indicates that there are 12456 bytes of unused program space left in memory, and 4567 bytes of unused data space left. The "OK" indicates the RUGID is ready to accept your next command.

Indirect commands consist of valid statements preceded with line numbers. Statements like

```
155 D3=17.5
250 PRINT A,B,C
```

are indirect commands. Indirect commands are stored for later execution rather than executed immediately. They, therefore, become the program that is executed whenever you type RUN. They are stored and executed in the numbered order indicated by the line numbers. If you enter a statement with a line number the same as one already in memory, the new statement will replace the one in memory. You should not do that because BASIC will clear all variables in that instance.

4.2 STATEMENTS

A statement consists of one or more commands ending with the RETURN or ENTER keystroke. The BASIC interpreter will begin execution with the first command on a line and will execute all the commands in a statement unless a command in the statement causes the interpreter to jump to another statement. If more than one command is to be executed on one statement, the commands must be separated with a colon. The following is a statement with three commands:

```
START X=1:Y=X+5:Z=X*Y
```

4.3 ENTERING AND EDITING PROGRAMS

There are two ways to enter and edit programs for RUGID. The first is to enter statements directly using a terminal, or using a computer that emulates a terminal. This direct manual entry method is OK for very small test programs. The second is to enter and edit the program on any computer with a word processor and serial port, and then download the program serially to RUGID. You will find the latter method more convenient for production programming.

4.3.1 ENTERING A PROGRAM USING A TERMINAL

Any ASCII terminal will work. See section 2.2 and 2.3 on connecting terminals and computers to RUGID's serial port. You must get RUGID to the command mode to begin editing programs. To do that, type "CTRL K" three times in succession. RUGID will go to the monitor mode and display the welcome prompt. Now type "C" to enter the command mode. In this mode you can edit the program and execute direct commands.

4.3.2 ENTERING A PROGRAM USING A COMPUTER

The second method of entering programs is to first enter the program into a computer using a word processor and then download the program to RUGID serially. See section 2.3 on connecting a computer to RUGID's serial port.

If your word processor has a nondocument mode, you should use that mode to edit your program, as that mode will not use imbedded non- ASCII codes. You have two choices regarding the format of the program you enter. You may enter it in a fairly strict BASIC line numbered syntax that will download directly to RUGID; or you may enter your program in a non-line numbered freeform format, and then run it through the supplied CONVERT.EXE program that converts it to the line numbered syntax. We use the latter method for all BASIC source programs as it relieves our programmers from the inconvenience of dealing with line numbers, and it enables the source code to be self documenting. All the example programs in this document use the non-line numbered form.

4.3.3 CONVERT.EXE

CONVERT.EXE is an executable utility for converting free form BASIC source code to line numbered format required by RUGID BASIC. The program runs on IBM PC compatibles with MSDOS/PCDOS 2.1 and later. It enables you to write your application source code in a more or less free form format with imbedded comments, mixed upper and lower case characters, blank lines, imbedded blanks and tabs, and, perhaps most importantly, without line numbers. CONVERT.EXE will read your source file and produce a compressed file with comments, blanks and tabs stripped out, and with line numbers inserted so that the syntax is acceptable to RUGID. The file produced by CONVERT will have the same name as your source file but with the .CMP extension. In order to eliminate line numbers in your source code you must use labels for each line to which you wish to GOTO or GOSUB. Adhere to the following rules and you should have no problems:

1. Variable names, instructions, and labels may be upper or lower case.
2. Blank lines are ignored.
3. A single quote mark signifies the start of a comment which is assumed to continue to the end of the line.
4. Strings inside double quote marks are left uncompressed.
5. Labels must be less than 10 characters long
6. Missing labels will be flagged as errors.
7. Duplicate labels are not flagged.
8. REM's are left in the program. So use REM if you wish the remark to stay in the program; or use a single quote if you wish the remark deleted in the version to be loaded to RUGID.

To invoke CONVERT to convert the source program DEMO.SRC to the compressed version DEMO.CMP, simply type in CONVERT DEMO.SRC. The resulting file DEMO.CMP will be compatible with RUGID BASIC syntax. The following example illustrates the source and final code segments produced by CONVERT. The code segment is a useful utility that reads the real time clock and produces a consistent time and date string in the variable J$ useful for time tagging alarm messages on the display or printout.

SOURCE CODE SEGMENT: File RTC.SRC

```
'**********Setup RTC string in J$:**********
 rem Notice that this rem stays in program
RTC I=CK%(6)
 if I<1 or I>7 then J$="    ":goto SKIPDAY
 I=3*(CK%(6)-1)+1:C0=CK%(1)
 J$=mid$("SUNMONTUEWEDTHUFRISAT",I,3)+" "              'Day of week

SKIPDAY I=4:gosub TIMEX :J$=J$+"/"                     'Month
 I=3:gosub TIMEX :J$=J$+"/"                            'Day of month
 I=5:gosub TIMEX :J$=J$+" "                            'Year
 I=2:gosub TIMEX :J$=J$+chr$(58)                       'Hours:
 I=1:gosub TIMEX :J$=J$+" ":return                     'Minutes

TIMEX A$=str$(CK%(I)) 'read clock & format to 2 chars.
 IF len(A$)<3 then J$=J$+"0"+right$(A$,1):return
 J$=J$+right$(A$,2):return
```

RESULTING CONVERTED VERSION: File RTC.CMP

```
1REMNOTICETHATTHISREMstays in program
```

```
2I=CK%(6)
3IFI<1ORI>7THENJ$="    ":GOTO6
4I=3*(CK%(6)-1)+1:C0=CK%(1)
5J$=MID$("SUNMONTUEWEDTHUFRISAT",I,3)+" "
6I=4:GOSUB110:J$=J$+"/"
7I=3:GOSUB110:J$=J$+"/"
8I=5:GOSUB110:J$=J$+" "
9I=2:GOSUB110:J$=J$+CHR$(58)
10I=1:GOSUB110:J$=J$+" ":RETURN
11A$=STR$(CK%(I))
12IFLEN(A$)<3THENJ$=J$+"0"+RIGHT$(A$,1):RETURN
13J$=J$+RIGHT$(A$,2):RETURN
```

4.4 SPECIAL CHARACTERS

The following characters have special meaning to BASIC:

CHAR	MEANING
@	Erases current line being typed. This is useful for deleting a line being entered that is known to contain an error since BASIC does not support line editing.
:	The colon is used to separate statements on the same line. There may be as many statements as desired on one line. Putting as many statements as possible on one line is recommended because it reduces storage requirements and statement searching time. A GOTO or GOSUB can only reference the beginning of a line because remaining statements on the line cannot have line numbers.
?	The question mark is used as a shorthand for PRINT, and may be used anywhere a PRINT instruction would otherwise be used. When a program is listed that used ? in place of PRINT, the word PRINT will be listed at each occurrence where ? was used in place of PRINT.
$	A dollar sign appended to a variable name establishes that the name is a character string. Also note that all instructions that end with a "$" are string instructions that produce string results.
%	A percent character appended to a variable name establishes that the variable is an integer with a range of -32768 to 32767.
,	Comma is used as a separator in PRINT and other statements. In a PRINT statement, it will cause the printing of values to occur on 14 character tabs.
;	Semicolon is also used as a separator in print statements and should be used for adjacent printing of values.
=	Equal symbol assigns the value to the right of the equal to the variable on the left of the equal.
+	Plus sign acts as the addition operator.
-	Minus sign reverses the arithmetic sign of the expression or variable that follows. It also acts as the subtraction operator.
*	The asterisk is the multiplication operator.

/	The slash is the division operator.
^	The caret is used to command exponentiation to integer powers.
(	The left parenthesis begins the grouping of an expression that is to be evaluated and then is to be treated as a term in an expression.
)	The right parenthesis terminates the expression that was started with a "(" above. A "(" must always have a matching ")".
^K	Holding the CONTROL key down and hitting the K key three times will stop a running program and return control to the monitor.
^T	Holding the CONTROL key down and hitting the T key three times will stop a running program and return control to BASIC command mode.

4.5 RESERVED KEYWORDS

The following words are reserved by BASIC as instructions so cannot be used in variable names:

ABS	FOR	NEXT	SAVE
AND	FRE	NOT	SGN
ASC	GET	NULL	SIN
ATN	GOSUB	ON	SPC
CHR$	GOTO	OR	SQR
CLEAR	IF	PEEK	STEP
CONT	INPUT	POKE	STOP
COS	INT	PRINT	STR$
DATA	LEFT$	POS	TAB
DEF	LEN	READ	TAN
DIM	LET	REM	THEN
END	LIST	RESTORE	TO
EXP	LOAD	RETURN	USR
FETCH	LOG	RIGHT$	VAL
FILE	MID$	RND	WAIT
FN	NEW	RUN	

4.6 VARIABLE NAMES

A variable name can be any alphabetic character (letters A to Z) followed by any number of alphanumeric characters (letters A to Z and numbers 0 to 9). Any alphanumeric characters after the first two will be ignored, so , for example, the variable names ABC and ABZ would be interpreted as the same variable. Such ambiguity will not result if you use one or two character variable names, and the program will run faster and take less RAM to store as well. Note that the variable with a single alphabetic character such as X or Y will be regarded as the same as X0 or Y0 respectively.

Variable names must use appended characters to differentiate floating point, integer, and string variables and arrays. The following table identifies the different ways of interpreting the variable A1 and the amount of storage required for each.

NAME	TYPE	# BYTES	RANGE
A1	Floating Point	7	+/-1.7E38 to 1.7E-39
A1%	Integer	7	-32768 to +32767
A1$	String	7+string	0 to 255 characters
A1(I)	Floating array	7+5*I	+/-1.7E38 to 1.7E-39
A1%(I)	Integer array	7+2*I	-32768 to +32767
A1$(I)	String array	7+3*I+strings	0 to 255 characters

4.7 ARRAYS

Arrays are variables that can reference several values in memory using a parenthesized subscript that follows the variable name to identify the particular element in the array that is to be stored or retrieved from memory. Array variables are also called subscripted variables because the variable name is always followed by the subscript. All arrays must be dimensioned using the DIM statement to specify how many elements the array is to have. For example, the statement DIM A(45),B%(14,4) will cause the BASIC interpreter to allocate space for 46 floating point numbers for the array A(), and 75 integer numbers for the array B%(). (Remember that A(0) is also valid.) The following are valid array specifications:

A(3)	Floating
A(I,5)	Floating, 2-dimensional
AX(I,J,K)	Floating, 3-dimensional
AX%(I,J)	Integer, 2-dimensional
AX$(5,J)	String, 2-dimensional

The variable name can be any valid variable name as identified above, but must not duplicate another variable name. The subscript within the parentheses may have up to 255 dimensions separated by commas. Each dimension may be a positive integer value of 0 to 32765 in the form of a number, a variable, or an expression. As many as 255 dimensions may be used, but in practicality, more than 3 are seldom used. Array variables encountered by BASIC when running a program that have not previously been dimensioned by a DIM statement will automatically be dimensioned by the interpreter upon first reference to the variable name to an array with 11 elements. Good programming practice requires that the programmer explicitly dimension each array in order to definitely establish the array size for future reference, and to avoid wasting memory in the form of unused array space.

Note that integer arrays make the most efficient use of memory of any number storage technique. If, for example, you want to store 100 numbers (say, 100 A/D converter samples), the following amounts of storage would be required:

Variable	Storage Required
AA..AZ BA..BZ CA..CZ DA..DZ	700 bytes
AA%..AZ% BA%..BZ% CA%..CZ% DA%..DZ%	700 bytes
AA(99)	507 bytes

AA%(99)	207 bytes

Therefore, when possible store numbers as integer numbers in integer arrays.

4.8 PROTECTED AND UNPROTECTED MEMORY USAGE

RUGID computers segment memory into protected and unprotected areas. When the BASIC system is in the command mode, i.e., when the program is being altered by the programmer, the protected memory is open and the program is stored there. When execution commences, as a result of keying in RUN or upon power application, the protected area will be closed so that inadvertent alteration to the program cannot occur. See section 6.B for the procedure for opening and closing memory. Consult the memory map in section 9.4 regarding areas of memory reserved and available for storage.

Reading data stored in the protected area by PEEKing into the area can be done at any time regardless of write protection.

4.9 PREASSIGNED VARIABLE NAMES

In order to simplify the programmer's job of interfacing to the A/D converters, relays, etc., software is provided to accomplish scanning and multiplexing of the I/O ports. The programmer need not access the ports directly, but only needs to store or retrieve data from integer variables having the names listed below in order to use the interrupt driven I/O scanning functions provided.

Variable Name	Function	Max # Elements
Arrays:		
AI%(I)	Analog inputs	75
AO%(I)	Analog outputs	32
AR%(I,J)	Data received from a remote unit	Depends on RAM
AT%(I,J)	Data to be sent to a remote unit	Depends on RAM
CK%(I)	Clock/calendar	7
DI%(I)	Digital inputs	144
DO%(I)	Digital outputs	144
DT%(I)	Digital timers	64
DU%(I)	Pulse duration outputs	48
DX%(I)	Digital input pulse counters	48
FW%(I)	Store & forward address path	16
PD%(I)	Pulse duration input analog values	8
P1%(I)	Data to be plotted to the LCD	256
P2%(I)	Data to be plotted to the LCD	256
P3%(I)	Data to be plotted to the LCD	256
Nonarrays:		
CC%	Cursor Column (LCD)	
CR%	Cursor Row (LCD)	
PO%	Port Identifier	
AA$	String to pass to remote using CRC com.	
Z$	String to be stored or retrieved from the RAM bank	

When using the array variables above, it is important to dimension the arrays only to the extent necessary to support the application, since the executive will sample the I/O ports up to the number identified in the array definition. Therefore, in the case of analog inputs, if only 3 analog inputs are required for the application but 8 are dimensioned (DIM AI%(8)), all 8 will be sampled resulting in a sample rate only 3/8ths that possible if a dimension of AI%(3) were used. Note also that the zeroth element in each array is usually unused (i.e., AO%(0), DI%(0)...). Therefore, the 16 digital inputs on the main board can be accessed using DI%(1)...DI%(8), and DI%(0) can be used by the programmer for other purposes.

4.A CONTROLLING THE WATCHDOG TIMER

All RUGID units are equipped with a hardware watchdog timer that, if allowed to time out, will reset the computer. Upon initialization of the user's BASIC program, background software commences to reset the watchdog 56 times per second as long as background software (I/O scanning, communications, etc.) continues to run normally. If background software fails to cycle, the watchdog will restart the program. In this mode, no attention to the watchdog is necessary by the BASIC program. This is generally sufficient watchdog safety for most applications. The watchdog can assume a BASIC watchdog function by executing a POKE 531,1. In this mode, the watchdog timer will reset the unit unless it is retriggered at least once every 2.3 seconds by BASIC executing a POKE 530,1. This statement should be included in the main application program loop and anywhere else a time consuming process might take linger than 2.3 seconds. Executing a POKE 531,0 will cause the interrupt process to constantly retrigger the watchdog timer independently of the BASIC program, thus effectively disabling the watchdog timer. In the monitor mode or command mode, the watchdog timer will be retriggered frequently by the monitor and executive software since in those modes it is assumed that the unit is not running an unattended application.

4.B LISTING THE PROGRAM

The LIST command allows you to examine a program, and to optionally print it out on the parallel printer port. The following uses of the LIST command give the indicated results:

Usage	Result
LIST	Lists the entire program
LIST 45	Lists line 45
LIST 14-110	Lists lines 14 through 110
LIST -250	Lists from the current line through line 250
LIST 45-	Lists from line 45 through end of program

Hitting ^P before invoking the LIST function will cause the listing to be directed to the parallel printer port as well as the display. Hitting ^Q will turn off the print feature. A listing may be interrupted by hitting ^T three times. When using the LIST command, the RUGID computer will present 22 lines on the display and will then wait for a [SPACE] keystroke before presenting another set of lines.

4.C BASIC FAULT TRAPPING

Hitting the "F" key in the monitor mode will cause the RUGID to display the last BASIC error message and when it occurred. The time and date of occurrence will also be shown. This capability is provided to enable you to detect errors that occur infrequently in unattended applications. The occurrence of a BASIC execution error will stop BASIC execution for 2 seconds in order that the error can be displayed. The program will then resume running from the beginning with the variable set and variable contents that were present at the time the error occurred. If this is not desired, a CLEAR instruction should be executed at the start of the program to erase all old data values.

4.D PROGRAM EXECUTION CONTROL

BASIC interprets program execution control commands as follows:

RUN	Causes the program to run from the beginning.
END	Stops the program and returns to the command mode. Asubsequent CONT direct command will resume execution at the statement following an END command in the program. As many END statements as desired may be inserted in the program.
STOP	Same as END above, but will print BREAK IN XXX when the STOP is executed, where XXX is the line number of the statement with the STOP command.
CONT	Resumes execution with the statement following a STOP or END command.

4.E ARITHMETIC

4.E.1 OPERATORS

The following operators can be used as indicated:

Symbol	Use
+	Addition, string concatenation
-	Subtraction, negation
*	Multiplication
/	Division
=	Assigns a value to a variable

4.E.2 RULES FOR EVALUATING EXPRESSIONS

The above operators can be used to perform mathematical computations and will be executed in the order of precedence beginning with operations of highest precedence and working down to operations of lowest precedence, as listed below. This means that divisions and multiplications will be performed before additions and subtractions. For example, 17 + 21 / 7 equals 20, not 5.43... When the interpreter encounters operations of equal precedence, the left most is executed first. Therefore, 5 - 7 + 8 is 6, not -10. You can always explicitly use parentheses to alter the order of execution. For example, (17 + 21) / 7 equals 5.43... The following list begins with operations of highest precedence and ends with those of lowest precedence. Operations on the same line have equal precedence.

1) () Expressions in parentheses are evaluated first.

2) Negation -Z, where Z can be an expression
3) * and / Multiplication and division

4) + and - Addition and subtraction

5) Relational
All below have equal precedence:
= Equal
<> Not equal
< Less than
> Greater than
=< or <= Less than or equal to
=> or >= Greater than or equal to

Logical operators:

6) NOT Logical and bitwise NOT, as in negation

7) AND Logical and bitwise AND

8) OR Logical and bitwise OR

Note that a relational expression can be used as part of any expression. Relational expressions will result in a result of True (-1) or False (0). Therefore, (7 = 5) = 0, (18 = 18) =-1, (45 < 55) = 0, (45 > 55) = -1.

4.E.3 ASSIGNING VALUES TO VARIABLES

The following commands assign values to variables. The DIM command, not discussed here initializes all array values to zero for numeric arrays, and to nulls for string arrays. If an array already exists when a DIM is executed, then the DIM does not zero the array.

LET Use: [LET] variable = expression

Example: LET X = A + 23.4, or X = A + 23.4

The value of the expression is assigned to the variable. The LET is optional and is generally omitted.

READ Use: READ variable,[variable],...

Example: 150 READ A,B,C%

Values will be read into the variables listed in the READ expression in the order that they are listed from the first DATA statement in the program and continuing with subsequent DATA statements until all variables in the READ statement have been read. If an attempt is made to read more data than there are DATA statements to supply, an out of data error will occur.

DATA Use: DATA item,[item],...

Example: 125 DATA 14.2,3.6,8

The DATA statement supplies data to READ statements in order to implement table lookups. See the READ statement above. Strings can be read from DATA statements, however, if you want to read strings with colons, commas, or leading blanks, you must enclose the string(s) in double quotes.

RESTORE Use: RESTORE

Example: 250 RESTORE

The RESTORE command resets the DATA statement pointer back to the first DATA statement in the program. The next READ statement executed following a RESTORE command will obtain its data from the first DATA statement in the program.

PEEK Use: PEEK (location)

Example: 159 CC = PEEK (J)

PEEK allows you to directly read the RUGID computer's memory locations. The parenthesized address must be in the range of 0 to 65535.

POKE Use: POKE location, byte

Example: 123 POKE J,L

POKE is used to write directly to locations in the RUGID computer's memory. The specified byte (L) is written into the address specified by J. The byte value must be in the range of 0 to 255; and the address must be in the range of 0 to 65535. CAUTION: Confine POKE operations to memory areas not reserved for the RUGID operating system or erroneous program operation may occur.

CLEAR Use: CLEAR

Example: CLEAR

CLEAR sets all arithmetic variables to 0, and all string variables to nulls. All GOSUB and FOR pointers and nesting are reset, and the data statement pointer is set to point to the first data statement in the program.

4.E.4 ARITHMETIC FUNCTIONS

ABS Use: ABS (expression)
Example: 450 X1 = ABS(X)

ABS computes the value of the expression in parentheses and returns the absolute value.

COS Use: COS (expression)

Example: 450 X1 = COS(X)

COS returns the cosine of expression X, which must be in radians. If X is in degrees, then use COS(X/57.2958).

DEF FNx Use: DEF FNvariable name (variable) = expression

Example: 220 DEF FNH(X) = X * 180 / 3.15927

DEF FNx is used to define a function of the parenthesized variable so that it need not be repeated frequently. Instead, once the function is defined, it may be invoked simply by including the function FNx in an expression where the larger function would otherwise be used. In the above example, the "H" is used to differentiate this function from others that we may define such as FNQ, for example. The "X" represents a variable that may be passed when the function is called. The result is, therefore, a function of X in this case.

FNx Use: FNvariable name (variable)

Example: X1 = FNH(X)

The function call FNx allows a user defined function to be called within an expression. This is much more convenient than using a GOSUB subroutine call. The preceding two examples would convert the value of X in radians to X1 in degrees.

EXP Use: EXP (expression)

Example: 340 X1 = EXP(X+5)
EXP implements exponentiation, i.e., the constant "e" (2.71828) is raised to the power contained in the parentheses. The parenthesized expression must not exceed 88.0296.

INT Use: INT (expression)

Example: 235 X1 = INT(3.9)

INT returns the largest integer less than or equal to the parenthesized expression. In the example above, the result would be 3.

LOG Use: LOG (expression)

Example: 240 X1 = LOG(X * 3.2)

LOG returns the natural (i.e., base e) logarithm of the parenthesized expression. In order to obtain the base Y logarithm of X use the formula LOG(X)/LOG(Y). For example, to obtain the common (i.e., base 10) logarithm of I use LOG(I)/LOG(10).

RND Use: RND (parameter)

Example: 400 X1 = (C-D)*RND(X)+D

RND returns a random number between 0 and 1. The parameter in parentheses determines the generation of random numbers. If the parameter is less than 0, a new sequence of random numbers is begun using the parameter as a seed. If the parameter = 0, the function will return the last random number generated. If the parameter is greater than 0, a new random number between 0 and 1 will be returned based upon the original seed. The example above will return a random number in the range of C to D as long as X is positive.

SGN Use: SGN (expression)

Example: 270 X1 = SGN(X)

The SGN function returns the sign of the parenthesized expression, or zero if the expression is equal to zero.

SIN Use: SIN (expression)

Example: 450 X1 = SIN(X)

SIN returns the sine of the expression and assumes that the expression is in radians. Note that if X is in degrees, you should use SIN(X/57.2958).

SQR Use: SQR (expression)

Example: 560 X1 = SQR(X)

SQR returns the square root of the parenthesized expression. Be sure that the expression is positive or an error will result.

TAN Use: TAN (expression)

Example: 238 X1 = TAN(X)

TAN returns the tangent of the parenthesized expression assuming the expression is in radians. Note that if X is in degrees, you should use TAN(X/57.2958).

4.E.5 DERIVED FUNCTIONS

The following functions may be implemented using the intrinsic BASIC functions and using the formulas shown below (P2 = Pi/2 = 1.5708):

```
SEC(X) = 1 / COS(X)
CSC(X) = 1 / SIN(X)
COT(X) = 1 / TAN(X)
ARCSIN(X) = ATN(X / SQR(1-X*X))
ARCCOS(X) = -ATN(X / SQR(1-X*X)) + P2
ARCSEC(X) = ATN(SQR(X*X-1)) + P2*(SGN(X)-1)
ARCCSC(X) = ATN(1 / SQR(X*X-1)) + P2*(SGN(X)-1)
ARCCOT(X) = -ATN(X) + P2
SINH(X) = (EXP(X) - EXP(-X)) / 2
```

COSH(X) = (EXP(X) + EXP(-X)) / 2
TANH(X) = 1 - EXP(-X) / COSH(X)
SECH(X) = 1 / COSH(X)
CSCH(X) = 1 / SINH(X)
COTH(X) = 1 + EXP(-X) / SINH(X)

4.F LOGICAL FUNCTIONS

AND Use: AND logical expression...
Also: expression AND expression

```
Example: 340 IF A<B AND C=D GOTO 120
Also: 350 X1 = A AND B
```

The AND operator is used in IF statements, as illustrated in the first example above, to require that two or more logical tests be true before proceeding with the action specified at the end of the IF statement. In the second example, variables A and B are converted to 16 bit signed two's complement integers and then bitwise ANDed to form a result. Variables A and B must be in the range of -32768 to +32767 or an error will result.

OR Use:OR logical expression...
Also: expression OR expression

```
Example: 230 IF A>B OR C=D GOTO 340
Also: 240 X1 = X OR 45
```

The OR operator is used, as in the first example, in IF statements to require that any one of two or more logical expressions be true before execution of the action part at the end of the IF statement will take place. It is also used, as in the second example, to compute the bitwise logical OR of two expressions. The two expressions are first converted to 16 bit integers, as in AND above.

NOT Use: NOT logical expression
Also: NOT expression

```
Example: 248 IF NOT W2 THEN 250
Also: 260 X1 = NOT X
```

NOT is used in logical expressions to invert the true/false sense of a result, as in the first example above. It is also used to bitwise logically invert a 16 bit integer as in the second example.

4.G BRANCHING AND LOOPS

FOR..NEXT Use:

```
FOR variable=expression TO expression [STEP expression]
.
.
NEXT [variable]
```

```
Example: 230 FOR I = 1 TO 7 STEP 2
```

.
:
300 NEXT I

The FOR..NEXT statement is used to implement a loop that is executed repetitively. On each repetition through the loop, the statements between the FOR and NEXT parts are executed. The loop begins with the variable initialized to the value computed in the first expression (1 in the example). When the NEXT is encountered, the variable is incremented by the value of the expression following the STEP part (2 in this example), and compared with the value of the expression following the TO part (7 in the example). If the variable is less than the final value (7), then the loop will be executed again; but if it exceeds the final value, execution will proceed with the first statement following the NEXT statement. Every FOR must have a NEXT. Also, you should never jump out of a FOR NEXT loop. If a loop needs to be terminated before its index value reaches its end value, then the index value should be set to a high value and the loop be allowed to terminate normally. Otherwise, a stack overflow could result.

GOSUB Use: GOSUB line number

Example: 460 GOSUB 500

GOSUB causes the interpreter to branch to the specified line number, which is the beginning of a subroutine. Execution proceeds beginning with that line number until a RETURN is encountered, at which time the interpreter returns to the statement immediately following the GOSUB. GOSUB's may be nested, i.e., a statement within a GOSUB may call another GOSUB. GOSUB's (i.e., subroutines) are useful for implementing routines that are to be used in several places in a program. The subroutine can be written once and then called whenever necessary.

GOTO Use: GOTO statement number

Example: 560 GOTO 700

The GOTO causes the interpreter to branch to the specified statement number, and proceed from there.

IF..GOTO Use:IF expression GOTO line number

Example: 320 IF X1 = 56 GOTO 340

In the IF..GOTO, if the expression is true, the interpreter branches to the specified line number; if not true, the interpreter branches to the next statement. Note: the IF must be the first command on the line.

IF..THEN Use: IF expression THEN statement [:statement]...

Example: 620 IF X2 = 78 THEN X1 = X2 + 6

The IF..THEN statement tests the expression. If it is true, the statement(s) following the THEN are executed; if not true, the interpreter proceeds to the next line of the program without executing the statement(s) immediately following the THEN. Note that IF statements must not be preceded on a line by any other statement or an erroneous error message may intermittently occur.

ON..GOTO Use:ON expression GOTO line number [,linenumber]...

Example: 10 ON I+J GOTO 25,35,45,55

The ON..GOTO is used to cause the interpreter to branch to any of several line numbers depending upon the value of the expression. If the expression = 1, the branch will be to the first line number specified; if the expression = 2, the branch will be to the second line number specified, etc. The expression must evaluate to the range of 0 to 255 or an error will occur. If it evaluates to zero or to a number greater than the number of line numbers specified, the interpreter will proceed to the statement following the ON..GOTO statement. As many line numbers as will fit on the line after the GOTO may be specified.

ON..GOSUB Use: ON expression GOSUB line number [,line number]...

Example: 340 ON I+5 GOSUB 35,45,55,65,75

ON...GOSUB is identical to the ON...GOTO above except that the interpreter will return to the first statement following the ON...GOSUB when it encounters a RETURN statement that terminates the target subroutine.

RETURN Use: RETURN

Example: 560 RETURN

The RETURN statement tells the interpreter when a subroutine is finished. Upon encountering a RETURN, the interpreter will proceed with the first statement following the previous GOSUB statement. There may be more than one RETURN statement in a subroutine, but there must be at least one.

4.H STRINGS

A string is a message that may consist of a collection of alphanumeric and special characters and that is treated as if it were data. A string variable (i.e., one that ends with $) can be set equal to a string of characters as in the following example:

C2$ = "RUGID COMPUTER"

The string must be in double quotes and may be from 0 to 255 characters long. The following commands enable you to concatenate, split, truncate, and convert string data. Note that commands that end with $ symbol produce string results; and those that do not, produce numeric results.

ASC Use: ASC(string expression)

Example: 670 X1 = ASC(C2$)

The ASC operation returns the numeric equivalent of the first character of the parenthesized string expression. See Section 10 for the ASCII to numeric conversion table. The string expression must not be the null (i.e., empty) string or an error will occur.

**CHR$** Use: CHR$(string expression)

Example: 340 C3$ = CHR$(13)

The CHR$ function returns the one character ASCII equivalent of the parenthesized expression. See Section 10 for an ASCII to numeric conversion table. The expression must evaluate to the range of 0 to 255 or an error will occur. The most common use of the CHR$ function is to cause control characters to be sent to a port. The example above would cause C3$ to be set equal to the ASCII code for carriage return.

**LEFT$** Use: LEFT$(string expression, length)

Example: 220 C3$ = LEFT$("RUGID COMPUTER",3)

The LEFT$ function extracts the leftmost characters from a string. The number of characters is determined by the length specification inside the parentheses. In the above example, C3$ would be set equal to "RUG".

LEN Use: LEN(string expression)

Example: 550 X1 = LEN("RUGID COMPUTER")

LEN returns the length of a string expression, i.e., the number of bytes in the string. All characters including special characters, punctuation, and blanks are included in the count. In the above example, X1 would be set equal to 14. (The enclosing double quotes are not counted.)

**MID$** Use: MID$(string expression, start [, length])

Example: C2$ = MID$("RUGID COMPUTER",7,4)

The MID$ function returns one or more characters from the specified start location within a string. The number of characters is specified by the optional length parameter. If the length parameter is omitted, all characters from the specified start character to the right hand end of the string will be returned. The start and length parameters must be in the range of 1 to 255. In the example, C2$ would be set equal to the string "COMP".

**RIGHT$** Use: RIGHT$(string expression, length)

Example: C2$ = RIGHT$("RUGID COMPUTER",5)

The RIGHT$ function returns the rightmost characters of string. The number of characters is determined by the length parameter. The length parameter must be in the range of 1 to 255 or an error will occur. In the example, C2$ would be set equal to "PUTER".

**STR$** Use: STR$(expression)

Example: 468 C2$ = STR$(234.56)

The STR$ function converts the value computed in the expression to an equivalent string. In the example, C2$ would be set equal to the string "234.56".

VAL Use: VAL(string expression)

Example: 440 X1 = VAL("14.9")

VAL returns the numeric equivalent of the parenthesized string expression. In the example, X1 would be set equal to 14.9.

Concatenation:

Strings may be concatenated, i.e., added together, simply by connecting them together using + symbols. For example, the following series of statements would set C2$ equal to "PLOTTING":

```
10 A$ = "PLO"
20 B$ = "TT"
30 C$ = "ING"
40 C2$ = A$ + B$ + C$
```

4.I INPUT/OUTPUT STATEMENTS

GET Use: GET string variable

Example: 140 GET C2$

GET reads the input buffer corresponding to the port specified by PO%. The options for ports to be specified by PO% are:

```
PO% Port
--- ----------------
0 Keyboard
1 RS232 terminal port
2 RS232 modem port
4 Speech synthesizer status
5 Modem/tone control
```

The port to be read must be specified by writing the appropriate port identifier into PO% before executing the GET statement. GET will return a single character if the port buffer is not empty. If the port is empty, the null string will be returned. An easy way to determine if a character was input is to test the length of the string variable using LEN. When asking the user for an input, be sure to flush the buffer before prompting for an input in case old data is already in the buffer. After executing the GET statement, the interpreter will proceed with the next statement whether or not data are present in the input buffer. Therefore, you will need to repeat the GET statement until a complete character string is received. You can continue with other operations while waiting for user inputs by interleaving GET statements with other operations in the program until a complete response is received from the user.

INPUT Use: INPUT ["prompt string";] variable [,variable]...

Example: 234 INPUT A,B,C$

INPUT requests data from the previously specified port (PO%=X), and will wait indefinitely for an input until all variables listed in the INPUT statement have been entered. For this reason, INPUT should not be used if cessation of data sampling and control cannot be tolerated. The GET command is generally more useful for real time applications. Variables entered must be separated by commas. The last value typed must be followed with a carriage return. The prompt string is optional and, if included, will be displayed upon execution of the INPUT statement. If a RETURN is typed in response to an INPUT statement, the BASIC interpreter will go to the command mode. Typing CONT after the program has been interrupted in this manner will cause the program to resume execution with the INPUT statement. Rather than use the print statement, we recommend that the KBTOX routine in the examples in Appendix B be used.

PRINT Use: PRINT expression [,expression]...

Example: 25 PRINT A,B;C$

PRINT will cause the interpreter to print the expressions specified to the port specified by PO%. Valid PO% values are:

PO%	Port
0	Onboard LCD display
1	RS232 terminal port
2	RS232 modem port
3	Parallel printer port
4	Speech synthesizer
5	Modem/tone control

The value of PO% must be specified before executing the PRINT statement. The list of values sent to the port will be terminated with a carriage return unless a comma or semicolon follows the list of expressions to be printed. If a semicolon separates two expressions, they will be printed adjacent to each other; if a comma separates two expressions, they will be tabbed to begin printing every tenth character position with spaces inserted as necessary. Expressions to be printed may be numeric or string expressions.

When printing to the onboard LCD display, the display will scroll up and line feed as any terminal would unless special characters are sent to position the cursor appropriately. Variables CC% and CR% control the next location to be written. In graphic mode, CC% has a range of 0 to 255 and specifies the display dot column to be written next. CR% has a range of 0 to 63 and controls the dot row to be written next. In character mode, CC% has a range of 0 to 39, and CR% has a range of 0 to 7.

4.K MISCELLANEOUS OPERATIONS

FRE Use: variable = FRE(0)

Example: D% = FRE(0)

FRE returns the number of bytes left in the write protected RAM array, i.e., the amount of program space available. The 0 is a dummy operand.

REM Use: REM any text

Example: 340 REM THIS IS A REMARK

The remark statement is used to insert an explanatory comment into the program. Remark statements are not executed. Statements should not follow REM statements on a line because the REM statement causes the interpreter to skip the rest of the line.

USR Use: variable = USR(operand)

Example: 440 DU = USR(14)

USR calls a user defined assembly language subroutine and transfers the operand in BASIC's floating point accumulator. The advantage of calling an assembly language routine is that it will generally execute much faster than the equivalent BASIC routine. Upon executing the USR instruction BASIC will fetch the starting location from memory locations 4 (LS) and 5(MS) and jump to it. The subroutine starting address must be contained in these addresses or erroneous operation will result. Use POKE to establish these values before executing the USR instruction. The subroutine must end with a RTS (return from subroutine) instruction. BASIC execution will then proceed with the statement following the USR instruction. As many parameters as desired may be passed between the user's subroutine and BASIC by using the PEEK and POKE instructions.

WAIT Use: WAIT address, mask [,invert]

Example: 330 WAIT A,B,C

WAIT is used to test a particular byte for one or more bits to assume a particular state. When executed, WAIT reads the contents of the specified address (A), exclusive OR's the contents with the optional invert byte (C) if present, and then AND's the result with the mask (B). It will continue to sample the byte in this manner until the result is nonzero, at which time the interpreter will proceed with the next statement. To test to see if a particular bit is a one, omit the invert operand and set the corresponding bit in mask to a one. To test for a zero bit, set the corresponding bit in both mask and invert. The following statement will wait until the most significant bit of byte location 1456 is a zero:

```
400 WAIT 1456,128,128
```

The WAIT statement should not be used if continuous realtime operation is required.

SECTION 5

5.0 CRC COMMUNICATIONS

The RUGID operating system provides background software that implements secure inter-unit communications using CRC-16 (16 bit Cyclic Redundancy Check) geometric code error checking as described in this section. All the message formats described work with port 2 using any of the hardware standards implemented.

5.1 GENERAL DESCRIPTION

In typical operation an initiating unit sends a message to a destination unit. The message may contain data intended for the destination unit, or it may be a request for the destination unit to send data back. The destination unit then responds appropriately and the interchange is finished. For the exchange to be regarded as successful, the initiating unit must receive a response to the initiating message to confirm that the message was accepted. Each message contains a CRC field to validate the accuracy of the message. If a message is received with an invalid CRC, the message is discarded and no reply is generated. If the CRC is correct, then the received data are placed in the receiving array, AR%(SN,I), where SN is the address of the sending station. Then, a reply is generated and sent to that station containing data in the receiving station's AT%(SN,I) array. This is purely a background function. The initiating station's BASIC program iniates the transmission by POKEing to a specific location. Background software transfers the contents of the AT%() array to the unit's output buffer along with header information and CRC security, which it calculates. BASIC then waits for a change in the received data (AR%()) in order to detect that a response was received with correct CRC.

5.2 CONFIGURING RUGID FOR CRC COMMUNICATIONS

When RUGIDs are delivered, they are set up for standard ASCII communications on both communications ports The modem port, port 2, can be configured for full CRC communications by altering two bytes: the configuration byte and the transmitter delay byte. Note that the two bytes reside in the RUGID voted, protected RAM and are replicated 3 times each. You must therefore install the two bytes in three locations each as indicated. These can be set using the monitor memory alteration capability, but it is much easier to load and run the supplied program DIAGNOSX.CMP. That program will prompt you for operating mode, transmitter delay, station address, baud rate, etc. using its modem port (port 2) configuration capability. In addition to setting the port mode and timing characteristics, your program must dimension one or more of the arrays indicated below, so that RUGID background software will have a place to store incoming data, and a place to get data that is to be sent.

BIDIRECTIONAL TRANSFER ARRAYS:

AR%(S,N)	Receive virtual analogs
AT%(S,N)	Transmit virtual analogs
	Where: S=station number, N=index of analog value

REGISTERS WITHIN RUGID TO CONTROL OR MONITOR CRC COMMUNICATIONS:

LOCATION	FUNCTION
$0251=593	80 Byte transmit buffer
$03D3=979	80 Byte receive buffer
$021D=541	Triggers CRC message transmission
$021E=542	Destination address of BASIC-triggered transmission
$04D0=1232	Acknowledgement: $FF=255=no reply 0=remote is running BASIC 1=remote is in command mode 2=remote is in monitor mode
$7F03=32515	Unit address

CONFIGURATION BYTE:

Location: $7F0B, $7F1B, $7F2B (=32523, 32539, 32555)

7	6	5	4	3	2	1	0
Diagnos: 0=OFF 1=ON	Spare	Diagnostic Port 00= LCD 01=Terminal 02=Modem 03=Printer		2W/4W 0=2W 1=4W	Spare	TLM Mode: 00=RUGID 01=CRC 02=ASCII	

Most commonly used settings: $00=ASCII communications
$09=CRC communications

TRANSMITTER DELAY BYTE:

Location: $7F0C, $7F1C, $7F2C (=32524, 32540, 32556)

7	6	5	4	3	2	1	0
# nulls preceding message, 0 to 3		Delay in 56ths of a second between assertion of transmitter key and audio tone and the sending of message.					

Most commonly used settings: $10=RUGID telephone modem system
$30=Slow radio system

5.3 INITIATING TRANSMISSIONS FROM BASIC

Once the RUGID is configured for CRC type communications, BASIC needs to control two bytes to implement transfer of entire arrays between itself and another unit. The following procedures should be followed:

5.3.1 BIDIRECTIONAL TRANSFERS

Included in operating system versions 3.2A and later are message formats that enable data to be included in both the outgoing and reply messages. CRC at the end of the message checks the entire message. Information to be sent must be loaded into array AT%(S,N); information received will be stored in array AR%(S,N), where S is the station address, and N is the index pointing to the two byte values to be transferred. Therefore, each station in a system must dimension two arrays, AT%(S,N) and AR%(S,N), to hold data to be sent and received, respectively. The dimension values S and N must each be less than 256.

The main difference between the RUGID communication protocol and others is that the RUGID system does not pre-define the contents of the data field in each message. The programmer has full freedom to define what type of data goes where in the format. Typically, we assign the first byte for bank selection and transmission flag; bytes 2 through 8 for status information (on/off flags, alarms, echoes of digital inputs, commands to output relays, HOA bits, etc.; and the remainder for analog information such as setpoints, tank levels, flows, totalizations, etc. Sometimes, we can combine a few bits of status data with an analog value in a single 16 bit word to save space. The important thing to realize is that its up to the programmer to decide the contents of the telemetry array; and therefore, imperative that he understand and establish early in the software design process the specific data to be transferred in each word of the telemetry arrays.

In a typical transfer, the initiating station will load the AT% array with outgoing data, and then trigger the transfer. The destination station will receive the data and store the data in its AR% array. It will then reply with the contents of its AT% array which the originating station will store in its AR% array. The originating station controls the number of words sent from the AT% array, and the number of words the destination station is to send back by writing those numbers in AT%(S,0) and AR%(S,0) respectively, representing the highest array index to be transferred (255 maximum). Values in AT%(S,0) and AR%(S,0) will not be transferred and will be unaffected by the transfers.

At most, 32 words (64 bytes) of data can be sent in any message. If AT%(S,0) or AR%(S,0) contains a value higher than 32, then the 32 words in the array up to and including the index specified in AT%(S,0) or AR%(S,0) will be transferred. See section 5.6 for message formats.

The following example assumes initiating station #3 wishes to transfer AT% words 1 through 13 to station #7's AR% array, and receive station #7's AT% words 5 through 37. In particular, the following transfer would take place:

```
INITIATING STATION #3                              DESTINATION STATION #7

Initiating
AT%(7,1) to AT%(7,13) ===========================> AR%(3,1) to AR%(3,13)

Reply
AR%(7,5) to AR%(7,37) <========================== AT%(3,5) to AT%(3,37)
```

To accomplish this, the following procedure would have to be followed in the program at station #3 (no action is required at station 7):

1. Write desired data to AT%(7,1) through AT%(7,13).

2. Write AT%(7,0)=13 to specify highest index to be sent out. Write AR%(7,0)=37 to specify highest index to be returned.

3. Poke destination unit's address in location $021E=542, i.e., execute POKE 542,7.

4. Poke message type $60=96 in location 541, i.e., execute POKE 541,96.

5. Monitor one entry in AR%(7,I) to detect a change. At RUGID, we use the MS bit of AT%(SN,1) to hold a transmit flag that the receiving BASIC program can detect and clear.

6. The data transferred from the destination station is now in AR%(3,5) through AR%(3,37).

5.3.2 STORE AND FORWARD OPERATION

In radio applications where some stations are not visible to others such as in long canal or pipeline systems or where some stations are located behind mountains or in canyons, it will be desirable to have certain RUGID units store and forward messages between units that otherwise cannot communicate. This can be implemented by installing station addresses specifying the path the message is to take in the preassigned array FW%(). The following FW%() array entries must be supplied for each attempted communication:

ARRAY ENTRY	FUNCTION
FW%(0)	Specifies how many stations involved (3 to 15)
FW%(1)	Source address (initiating station)
FW%(2)	Address of first station to forward message
FW%(3)	Address of second station to forward message
.	.
.	.
.	.
FW%(N)	Destination station address

In operation, array elements FW%(1) through FW%(N) are installed by background software in the message to specify the path before the initial transmission. Each receiving station checks CRC and then checks if the message is destined for it. If so, it then checks if its address is contained in the path. If so, the station will transfer the message from its receive buffer to its transmit buffer, install its own address as the source address, recalculate CRC, and retransmit it. The path is followed in reverse for the reply messages. Note that even if all stations can successfully receive all messages, the message will still follow the specified path due to the method of handling address security within each unit. This is a powerful store and forward implementation because the initiating station specifies the path and makes the ultimate decision whether the message was transferred successfully. Therefore the initiating station can specify an alternate path and retry the message if a succession of failed messages occurs with the initial path.

5.3.3 TEXT TRANSFERS

Strings can be transferred to remote stations using bidirectional transfer message type $70 (112). In this scheme, the Basic program writes the string into string variable AA$, specifies the destination station address, then triggers the transfer by poking the message type. The following program segment would send the string "Hello RUGID station" to remote station 13:

```
100 AA$="Hello RUGID station"        'Load the string
200 POKE 542,13                      'Specify the remote address
300 POKE 541,112                     'Trigger the transfer
```

At the remote station, the string would appear in the port 1 input buffer as if it had arrived from a local terminal. This technique can be used to send messages to remote sites to await personnel when they arrive, or to pass display information. It can also be used to delete and reload the program at the remote site.

5.3.4 MESSAGE FORMATS, BIDIRECTIONAL TRANSFERS:

	HEADER						
Byte	0	1	2	3	4	5	6
Function	ID	Sender Address	Receiver Address	Message Type	High Transmit Index	High Reply Index	Message Length
Example	$83	$03	$04	$64	$0D=13	$25=37	$1B=27

	Forward Path	DATA FIELD	Security
Bytes	7-10	11-24	25,26
Function	Fwd Path	AT%(7,1) through AT%(7,7)	CRC-16
Example	3,4,5,7	00,01,00,02,00,03,00,04,00,05,00,06,00,07	CRC-16

REPLY:

	HEADER						
Byte	0	1	2	3	4	5	6
Function	ID	Sender Address	Receiver Address	Message Type	Hi Transmit Index	Error Reply	Message Length
Example	$83	07	05	$E4	$25=37	00	$4D=77

	Forward Path	DATA FIELD	Security
Bytes	7-10	11-75	76,77
Function	Fwd Path	Destination's AT%(1,5) through AT%(1,37)	CRC-16
Example	3,4,5,7	00,05,00,06,00,07......00,37	CRC-16

EXPLANATION:

Initiating station #3 sends AT%(7,1) through AT%(7,7) to receiving station #7 where the 13 words transferred are stored in AR%(3,1) through AR%(3,7). Destination station #7 replies by sending words AT%(3,5) through AT%(3,37) to station #3 where the 32 words are stored in AR%(7,5) through AR%(7,37). The values shown in header bytes correspond to those in the example in section 5.3.3. Addresses The forward path specifies that the message be transfered from station 3 to 4 to 5 to 7. To accomplish this forwarding, array FW%() must have the following values:

FW%(0)=4	(4 stations total in communication)
FW%(1)=3	(Initiating station is address #3)
FW%(2)=4	(First forwarding site)
FW%(3)=5	(The next forwarding site)
FW%(4)=7	(The destination station address)

5.4 MODBUS FORMAT MESSAGES

All RUG6 units with operating system versions 3.2L and later support MODBUS format messages ON PORT 1. This mode is intended to enable a RUG6 unit, acting as a master communications controller in a system of multiple RUGID's, to pass information between a PC master computer and the field RUGID's. Communications between the master RUGID and the remote RUGID's would still use the CRC secured formats described above, and data would still be passed between AT% and AR% arrays. However, communications between the master RUGID and a PC computer would use the MODBUS protocol. This enables any of a number of software packages that support the MODBUS protocol to act as PC masters to RUGID systems. This mode of operation is invoked by setting bit 2 of locations $7F0F, $7F1F, and 7F2F to a "1"; i.e., if these locations were to have a content of $40 (usually the case) then they should be set to $42.

RUG6 units with operating system versions 3.4B and later support longer MODBUS messages (up to 255 bytes/message) and allocate more telemetry space per field RTU (256 bytes per RTU) by using a RAM bank in the master RUG6 to hold the field data base. The RAM bank consists of a single 32K board with the first chip installed and allocated for telemetry use with a single 256 byte page used for each field RTU and 128 pages total. This mode is enabled by turning on bits 2 and 4 in locations $7F0F, $7F1F, and $7F2F. See paragraph 5.7.2 below.

Timing of MODBUS messages is handled by background software. Basically, the RUGID acts as a slave to the PC master, responding when polled. Since the MODBUS format has no sync byte, the RUGID will resynchronize whenever a gap is seen in the data received from the PC. The time of the required gap for resynchronization is set in memory location $022B=555 in 56ths of a second. The allowed range is 1 to 255. Upon boot up, the register will be set to 56 for a default delay of one second. Therefore, to set the delay for anything other than one second, the BASIC program must set the byte after each boot up. To set the delay to 2 seconds, BASIC would execute the following:

```
POKE 555,112
```

5.4.1 MODBUS MESSAGES SUPPORTED

RUGID units support the following MODBUS messages:

MSG TYPE	STANDARD MODE	RAM BANK MODE	DESCRIPTION
1	*		Read output status
3	*	*	Read I/O registers
4	*	*	Read I/O registers
5	*		Preset single coil
6	*		Preset single register
16	*	*	Preset multiple registers

5.4.1.1 MESSAGE TYPE 1...READ OUTPUT STATUS

Message type 1 is used to read specific bits in the AT%() array as if they were phisical outputs. The format is the following:

Byte	0	1	2	3	4	5	6	7
Function	Station Address	Message Type	Data Start		Number of Bits to Read		CRC-16 Security	
Example	$05	$01	$00	$01	$00	$28	$XX	$XX

REPLY:

Byte	0	1	2	3	4	5	6	7	8,9
Funct	Sta. Addr	MSG Type	# Bytes	Bits 1-8	Bits 9-16	Bits 17-24	Bits 25-32	Bits 33-40	CRC-16
Val	05	01	05	01	02	03	04	05	XX,XX

5.4.1.2 TYPES 3 and 4...READ MULTIPLE REGISTERS

RUGID units respond identically to MODBUS commands 3 and 4 to read multiple registers. Basically, these messages enable the PC to read data received from a remote RUGID unit (the master's AR%(S,N) array), or to read data sent to a remote RUGID (the master's AT%(S,N) array). If the PC requests to read registers numbered lower than 255, then RUGID will respond with the contents of the AR%() array (received from a remote site). If the PC requests to read registers numbered higher than 256, then 256 will be subtracted from the register address, and the contents of the resulting registers from the AT%() array (transmitted to a remote site) will be returned to the PC. The following examples illustrate the request/response dialog:

READ AR%(5,2 to 8)...Read station 5 received registers 2 through 8:

REQUEST:

Byte	0	1	2	3	4	5	6	7
Function	Station Address	Message Type	First Register to Read		Number of Registers to Read		CRC-16 Security	
Example	$05	$03	$00	$02	$00	$07	$A4	$4C

REPLY:

Byte	0	1	2	3,4	5,6	7,8	9,10	11,12	13,14	15,16	17,18
Funct	Sta. Addr	MSG Type	# Bytes	Reg 2	Reg 3	Reg 4	Reg 5	Reg 6	Reg 7	Reg 8	CRC-16
Val	05	03	0E	00,01	00,18	00,0E	00,1D	00,06	00,5A	00,06	78,85

READ AT%(5,2 to 8)...Read station 5 transmit registers 2 through 8:

REQUEST:

Byte	0	1	2	3	4	5	6	7
Function	Station Address	Message Type	First Register to Read		Number of Registers to Read		CRC-16 Security	
Example	$05	$03	$01	$02	$00	$07	$A5	$B0

REPLY:

Byte	0	1	2	3,4	5,6	7,8	9,10	11,12	13,14	15,16	17,18
Funct	Sta. Addr	MSG Type	# Bytes	Reg 2	Reg 3	Reg 4	Reg 5	Reg 6	Reg 7	Reg 8	CRC-16
Val	05	03	0E	00,02	00,03	00,04	00,05	00,06	00,07	00,08	FA,15

5.4.1.3 MESSAGE TYPE 5...PRESET SINGLE COIL

This message is used to set or clear a single pseudo output; i.e., a single bit in the AT%() array.

REQUEST:

Byte	0	1	2	3	4	5	6	7
Function	Station Address	Message Type	Coil to Preset MS,LS		ON or OFF $00 or $FF	Spare (always zero)	CRC-16 Security	
Example	$05	$05	$00	$15	$00	$00	XX	XX

REPLY (Same as request):

Byte	0	1	2	3	4	5	6	7
Function	Station Address	Message Type	Coil to Preset MS,LS		ON or OFF $00 or $FF	Spare (always zero)	CRC-16 Security	
Example	$05	$05	$00	$15	$00	$00	XX	XX

5.4.1.4 MESSAGE TYPE 6...PRESET SINGLE REGISTER

This message is used to preset a value into a single 16 bit register. The message example below sets AT%(5,21) to a value of 7.

REQUEST:

Byte	0	1	2	3	4	5	6	7
Function	Station Address	Message Type	Register to Preset MS,LS		Data Value to Send MS,LS		CRC-16 Security	
Example	$05	$06	$00	$15	$00	$07	XX	XX

REPLY (Same as request):

Byte	0	1	2	3	4	5	6	7
Function	Station Address	Message Type	Register to Preset MS,LS		Data Value to Send MS,LS		CRC-16 Security	
Example	$05	$06	$00	$15	$00	$07	XX	XX

5.4.1.5 MESSAGE TYPE 16...WRITE MULTIPLE REGISTERS

In order to write to multiple registers, the PC issues a command that specifies RTU address, starting register number, number of registers to be written, and contents of each register. RUGID then stores the data in the addressed AT%(S,N) array entries. The following example illustrates the process:

WRITE values 3, 4, 5 and 6 to AT%(1,3 to 6):

REQUEST:

Byte	0	1	2,3	4,5	6	7,8	9,10	11,12	13,14	15,16
Funct	Sta. Addr	MSG Type	1st Reg to Write	# Regs to Write	# Data Bytes	Reg 3	Reg 4	Reg 5	Reg 6	CRC-16
Val	01	10	00,03	00,04	08	00,03	00,04	00,05	00,06	E0,BD

REPLY:

Byte	0	1	2	3	4	5	6	7
Function	Station Address	Message Type	First Register Written		Number of Registers Written		CRC-16 Security	
Example	$01	$10	$00	$03	$00	$04	$31	$CA

5.4.2 MODBUS MESSAGES USING MASTER RAM BANK

Operating system versions 3.4 B and later support the use of chip one of the RAM bank to store telemetry data retreived from and destined for field RTU's instead of using the AT%() and AR%() arrays. This frees up substantial RAM in the master unit, and allows the master to interface with up to 128 field RUG6's. Each field RUG6 is allocated one page (256 bytes) for telemetry storage within the RAM bank. The page number corresponds to the field RTU address, i.e., page 3 is used for RTU address number 3. Also, the first byte of the MODBUS message must now match the address of the master unit (usually 1); and the RTU slave address is now contained in the "first register" field of the MODBUS message. With this design, up to 256 RUG6's can be connected to a single MODBUS master, using the first message byte to address these units; and each RUG6 can have up to 128 RUG6 slaves communicating using the RUGID CRC secured format.

5.4.2.1 RAM BANK ORGANIZATION FOR MODBUS USE

Each 256 byte RAM bank page is organized as illustrated below. Basically, the page is organized as 128 words with the first collection of words used for outgoing data to a field RTU, and the remainder used for data that have been received from a field RTU. The last three bytes of the page contain special information as follows:

Byte $FD=253	Number of words to transmit.
Byte $FE=254	First word of receive field in RAM bank.
Byte $FF=255	Number of words to recieve.

EXAMPLE: One page allocating 11 words for transmit, 33 words for receive, and specifying word 39 as the first word to hold received data.

```
<==============32 bytes, 16 words=======================>

00<==========TX these words (1 to 11)==========>12..............>
16........................................................................................................>
32...................38<==========================================
48============Receive these words (39 to 72)==============
64=====================================>73..............................
80.........................................................................................................
96.........................................................................................................
112...................................................................................... 11 39 33
```

5.4.2.2 TYPES 3 and 4...READ MULTIPLE REGISTERS

RUGID units respond identically to MODBUS commands 3 and 4 to read multiple registers. Basically, these messages enable the PC to read data received from a remote RUGID unit (in the master's receive area in the RAM bank), or to read data sent to a remote RUGID (the master's transmit area in the RAM bank). The following examples illustrate the request/response dialog:

Read station 5 registers 2 through 8:

REQUEST:

Byte	0	1	2	3	4	5	6	7
Function	Station Address	Message Type	First Register to Read		Number of Registers to Read		CRC-16 Security	
Example	$01	$03	$05	$02	$00	$07	$XX	$XX

REPLY:

Byte	0	1	2	3,4	5,6	7,8	9,10	11,12	13,14	15,16	17,18
Funct	Sta. Addr	MSG Type	# Bytes	Reg 2	Reg 3	Reg 4	Reg 5	Reg 6	Reg 7	Reg 8	CRC-16
Val	01	03	0E	00,01	00,18	00,0E	00,1D	00,06	00,5A	00,06	XX,XX

5.4.2.3 MESSAGE TYPE 16...WRITE MULTIPLE REGISTERS

In order to write to multiple registers, the PC issues a command that specifies RTU address, starting register number, number of registers to be written, and contents of each register. RUGID then stores the data in the addressed RAM bank area. The following example illustrates the process:

WRITE values 3, 4, 5 and 6 to station 5 registers 9 through 12:

REQUEST:

Byte	0	1	2,3	4,5	6	7,8	9,10	11,12	13,14	15,16
Funct	Sta. Addr	MSG Type	1st Reg to Write	# Regs to Write	# Data Bytes	Reg 3	Reg 4	Reg 5	Reg 6	CRC-16
Val	01	10	05,09	00,04	08	00,03	00,04	00,05	00,06	XX,XX

REPLY:

Byte	0	1	2	3	4	5	6	7
Function	Station Address	Message Type	First Register Written		Number of Registers Written		CRC-16 Security	
Example	$01	$10	$00	$09	$00	$04	$XX	$XX

Whenever the master unit writes to one or more registers using the above message, the addressed RTU will be indicated in location $0220 (544).

SECTION 6

6.0 HOWTO...

In this section you will see how to do some things that you may not find specifically addressed elsewhere in this manual. Such topics as controlling peripherals, changing batteries, etc. are discussed.

There are three ways to control or read any peripheral on the main or expansion board. One is to use the predefined variables via BASIC; another is to use PEEK or POKE, also from BASIC. The third is to write an assembly language routine that accesses the peripheral directly. This third method is usually considerably more involved than accessing the port from BASIC, but gives faster control. This manual does not address assembly language programming.

6.1 TURN ON A DIGITAL OUTPUT (RELAY)

The main field terminal board is equipped with relays capable of controlling AC or DC loads. To control any of these digital outputs, it is simply necessary to set the corresponding entry in array DO%(N) equal to 0 to turn it off; equal to 1 to turn it on, and equal to 2 to make it flash on and off every half second, where N specifies the output channel to be controlled. For example, to turn on relay 3 and turn off relay 4, the following statement would be required:

```
230 DO%(3)=1: DO%(4)=0
```

When controlling the status outputs from assembly language programs, two precautions are necessary. First, the output latch is a UCN5801 octal latch/driver which may be written but not read. Therefore you must keep a copy of the latest byte written to the device in order to know what states are in effect. This is important in cases where you want to alter the state of one bit without effecting the others. The second precaution is related to the first. If you are controlling one or more bits of a digital output from an assembly language routine, you must not control others from BASIC. This is because BASIC keeps its own copy of the port byte, the entirety of which will be written to the port whenever any bit is written.

Note that the different RUGID models have differing complements of base relay capability, and employ unused relay drivers for other functions as defined in the table below.

DO%() CHANNEL	RUG6	RUG7	RUG8
DO%(1)	RELAY #1	RELAY #1	RELAY #1
DO%(2)	RELAY #2	RELAY #2	RELAY #2
DO%(3)	RELAY #3	RELAY #3	RELAY #3
DO%(4)	RELAY #4	RELAY #4	RELAY #4
DO%(5)	RELAY #5		ON=puts unit to sleep
DO%(6)	RELAY #6		
DO%(7)	RELAY #7	ON=Enable hi rate battery charge	
DO%(8)	RELAY #8	ON=turn on display backlight	

6.2 FLASH A DIGITAL OUTPUT ON AND OFF REPEATEDLY

For alarm annunciation, you may want to flash a digital output repeatedly. To do this you must simply write a 2 to the output. For example, to flash digital output #4, the following statement would be required:

```
240 DO%(4)=2
```

The output will flash until a different value is written to the output. Note that if you execute the statement DO%(8)=2 on a RUG7, it will cause the display backlighting to flash on and off.

6.3 READ A STATUS INPUT

There are 8 to 16 status inputs on the main circuit board and 32 on each expansion board. Any may be read by reading the corresponding entry in the digital input array DI%(N), where N specifies which input is to be read. For example, the following statement reads digital inputs 7 and 12 and puts the results into variables A and B respectively:

```
400 A=DI%(7): B=AI%(12)
```

When read, a 0 means that the input is high or open; a 1 means the input is low. A closed contact will produce a 1.

6.4 READ AN ANALOG INPUT

Analog inputs may be read by accessing BASIC array AI%(N), where N specifies the channel to be read. The 10 bit A/D converter on the main board produces values of 0 to 1023 when the input voltage spans 0 volts to +5.0 volts. Whenever the array AI%(N) is defined, monitor software will scan the analog inputs, applying one at a time to the A/D converter. The A/D converter output is then stored in the AI%(N) array. This process is interrupt driven and is transparent to and asynchronous with any other process. We do not advise that you attempt to control the A/D converter using assembly language due to the complexity of the multiplexing and interrupt interface tasks. As an example of how to read the analog values, the following BASIC statement reads analog inputs 1 and 5 and stores them in variables A and B respectively:

```
290 AI%(0)=4372                      'Sets AI's for 1 sample per second
295 A=AI%(1): B=AI%(5)
```

6.5 CONTROL AN ANALOG OUTPUT

Analog outputs are multiplexed in a manner similar to the analog inputs, so control by assembly language routines is not recommended. Control through BASIC is quite convenient, however, requiring only that you write the desired output value in the range of 1 to 4095 to an array entry AO%(N), where N is the channel number you wish to control. For example, if you wish to have analog output 2 put out a value of 20% of full scale, the following statement would suffice:

```
220 AO%(2)=.2*4095
```

6.6 PROGRAM A TIME DELAY

Time delays are almost always required in control applications to establish sample intervals, cycle equipment, etc. There are at least 3 ways to generate time delays using the RUGID computer. The first and least desirable is to use a software loop such as a FOR...NEXT loop with a large target value to generate the delay. This is undesirable because no other processing can be done while the loop is running. Two more desirable ways are to use the preassigned timer array DT%(N), or to use the real time clock.

Once per second the interrupt process examines the array DT%() for entries not equal to zero. If any are found, they are decremented by one. The decrementing ceases when the value in the register reaches zero. Therefore, you can set a time delay into one or more of the entries in DT%(N) and test for when it reaches zero. When it does, the time delay has passed.

An alternative way is to read the real time clock, add a value to it equal to the delay you want, and store that value in an array. Every time through the program cycle compare the stored value with the real time clock value, and take action when the real time clock value exceeds the stored value. This is much less convenient than using DT%(N), so that method is recommended. A more practical use of the real time clock is to trigger actions at a particular time of day. For example, if you want to trigger a printed report on each hour, the following statements would cause a jump to a printing subroutine (PRNT) on each hour:

```
.
.
IF CT%<>CK%(2) THEN CT%=CK%(2):GOSUB PRNT
.
.
```

In the above statement, CT% contains the present hour value. When the real time clock hours value (CK%(2)) changes, the printout commences. Note that the statement is part of a larger program that makes this test as it passes through. In general, every program will operate in this fashion because it must do many things seemingly at once, and, therefore, cannot stop to dwell on any single test. The use of the real time clock is much preferred to using DT%(N) in this instance because DT%(N) would accumulate a clock drift if BASIC did not sample it at least once per second. Using the real time clock eliminates this problem because the clock continues running after the time setpoint is reached.

6.7 HOOK UP A 4-20 MA. TRANSDUCER

Many industrial applications rely upon the 4-20 ma. current loop standard for transmission of analog information. This standard has the advantage that the analog measurement is transmitted as a current rather than as a voltage, making it less susceptible to noise and making it independent of the impedance in the wires over which the measurement is transmitted. The way it works is quite simple. A DC power supply sources voltage to the transducer over the positive wire to the transducer. The transducer is designed to act as a current valve, adjusting the current that it will let pass in proportion to the analog value it is measuring. The negative wire from the transducer is connected to one of the analog input ports on the RUGID computer, causing the loop current to pass through the input voltage divider resistors in the input circuit. When a current passes in the loop, the bottom resistor converts it to a voltage which is fed to the A/D converter where it is converted to a number. Connecting the RUGID computer's analog common to the power supply negative terminal completes the loop. Figures 3.0A and 3.0B illustrate the necessary connections. The power supply voltage should satisfy the following relationship:

Volts > 4 + (.02 * Loop Impedance) + Transducer Voltage

The required transducer voltage is usually in the range of 12 to 24 volts. One advantage of the current loop technique is that other instruments, such as analog recorders, meters, etc. can be inserted in series with the loop without degrading the accuracy of the measurement as long as the selected loop power supply voltage accounts for the additional voltage drops. Twisted shielded pair wire should be used with the shield tied only to the RUGID computer's analog common.

6.8 READ AND SET REAL TIME CLOCK VALUE FROM PROGRAM

The real time clock/calendar may be written or read at any time from BASIC simply by accessing array CK%(N). Writing to CK%(N) causes the processor to download the entire contents of CK%(N) to the clock/calendar integrated circuit. Since the clock/calendar chip remains powered during a power outage, time and date information will not be lost during an outage. The clock/calendar information is used by the monitor to record time of occurrence of BASIC errors for the "F" function. As an example of real time clock accessing, the following software computes the length of power outages to the unit:

In the initialization program:

```
.
.
50 PZ=CK%(2)+60*CK%(1)+3600*CK%(0)-PP      ' compute outage length
55 IF PP<0 THEN PP=PP+24
.
.
```

In the cycling program: .

```
.
500 PP=CK%(2)+60*CK%(1)+3600*CK%(0)        'save current clock value
.
.
```

In the above statements, PP keeps track of the current time of day in hours; PZ contains the length of the last power outage. Additional statements would be necessary to keep track of outages exceeding 24 hours.

6.9 HOOK UP AN EXTERNAL BATTERY BACKUP

RUG6 and RUG7 units are equipped with battery chargers to facilitate battery backup applications. To connect a battery, simply hook its positive and negative terminals to the BATT+ and BATT- terminals on the RUGID's field board, respectively.

6.A CHANGE ONBOARD LITHIUM BATTERY

The onboard Lithium battery can power the RAM and real time clock for approximately 2 years of power outage. The battery does not discharge during the time the unit is powered up. In order to change the onboard Lithium battery, it is necessary to remove the main circuit board from the unit, replace the battery, then reassemble the unit. Be sure that you have a type BR 2325 or equivalent coin type battery on hand for replacement. Follow this procedure:

1. Remove power

2. Remove unit top cover

3. Remove ribbon cables from main board (front board in housing).

4. Pull circuit board out and set on NONCONDUCTIVE surface with component side up.

5. Locate the large coin-shaped battery in a black plastic holder near the front of the main circuit board.

6. Remove the battery from the holder and replace with the new one, positive terminal up. You may have the battery out of the unit for up to 2 minutes without loss of program or real time clock data.

7. Place the main circuit board back in the unit and replace the cover.

8. Reapply power and verify that the program and real time clock contents are intact.

6.B REPLACE FUSE

The fuse is a 3AG 2 amp standard blow type mounted in soldered on clips on the main field terminal board. There is no fuse on the main board. In normal operation, the fuse should never blow. Therefore, you should ascertain and correct the problem that lead to the fuse blowing before replacing it. Follow the procedure below for replacement:

1. Disconnect AC power.
2. Remove fuse.
3. Replace with 3AG 2 amp fuse.

6.C DISPLAY LAST BASIC ERROR MESSAGE

Because errors can occur infrequently and often at times when the unit's operation is unobserved, the RUGID computers store the last error and time and date of occurrence. To examine the last error to occur you must exit the program and enter the monitor by hitting ^K^K^K. Then hit the F key and the unit will list the last error and time of occurrence.

6.D CONTROL MEMORY WRITE PROTECTION

Write protection is present in order to protect the program and critical data from inadvertent erasure due to program bugs or transients. It is possible to run the unit with write protection continuously removed, but one occurrence of a program erasure will convince you of the necessity of this feature. The RUGID computer's write protection is implemented using a 2.4 second time delay that must expire before the memory may be written. The following routines remove and install write protection:

To open RAM:

```
100 A=PEEK(969) AND 191:POKE 969,A:POKE 34800,A
(Wait 3 seconds, then RAM will be open.)

100 A=PEEK(969) OR 64:POKE 969,A:POKE 34800,A
(RAM will be closed immediately.)
```

6.E AUTODIAL A TELEPHONE NUMBER

To dial a phone number, you must:

1) Go off hook
2) Wait for dial tone
3) Dial the number

The following code segment accomplishes these tasks:

```
100 PO%=5                              'Setup to use port 5
110 PRINT "L"                   'Off hook
120 DT%(1)=3                    'Start dial tone delay
130 IF DT%(1)<>0 GOTO 130       'Wait for dial tone
200 PRINT "5551234"             'Dial the number
210 DT%(1)=2                    'Start dialing timer
220 IF DT%(1)<>0 GOTO 220       'Wait
210 RETURN
```

6.F HOOKUP RUGID TO A RADIO

RUGID computers are specifically designed for easy integration with standard telemetry radios. Figures 6.F.1, 6.F.2, and 6.F.3 present the connections to a standard Motorola radio. Only six wires, audio in, audio out, and transmitter key are required to accomplish the connection.

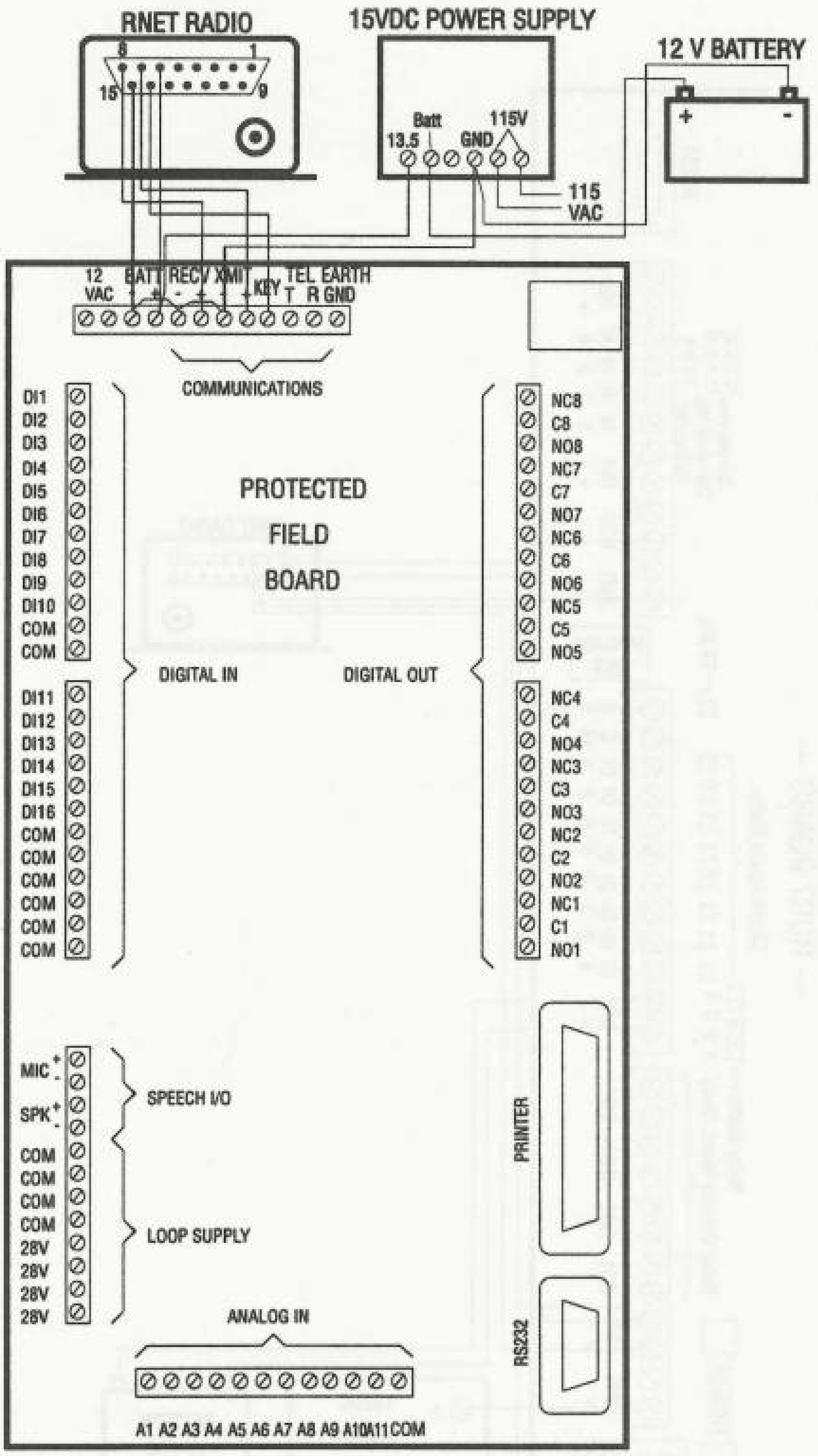


FIGURE 6.F.1 TYPICAL RUG6 RADIO CONNECTION

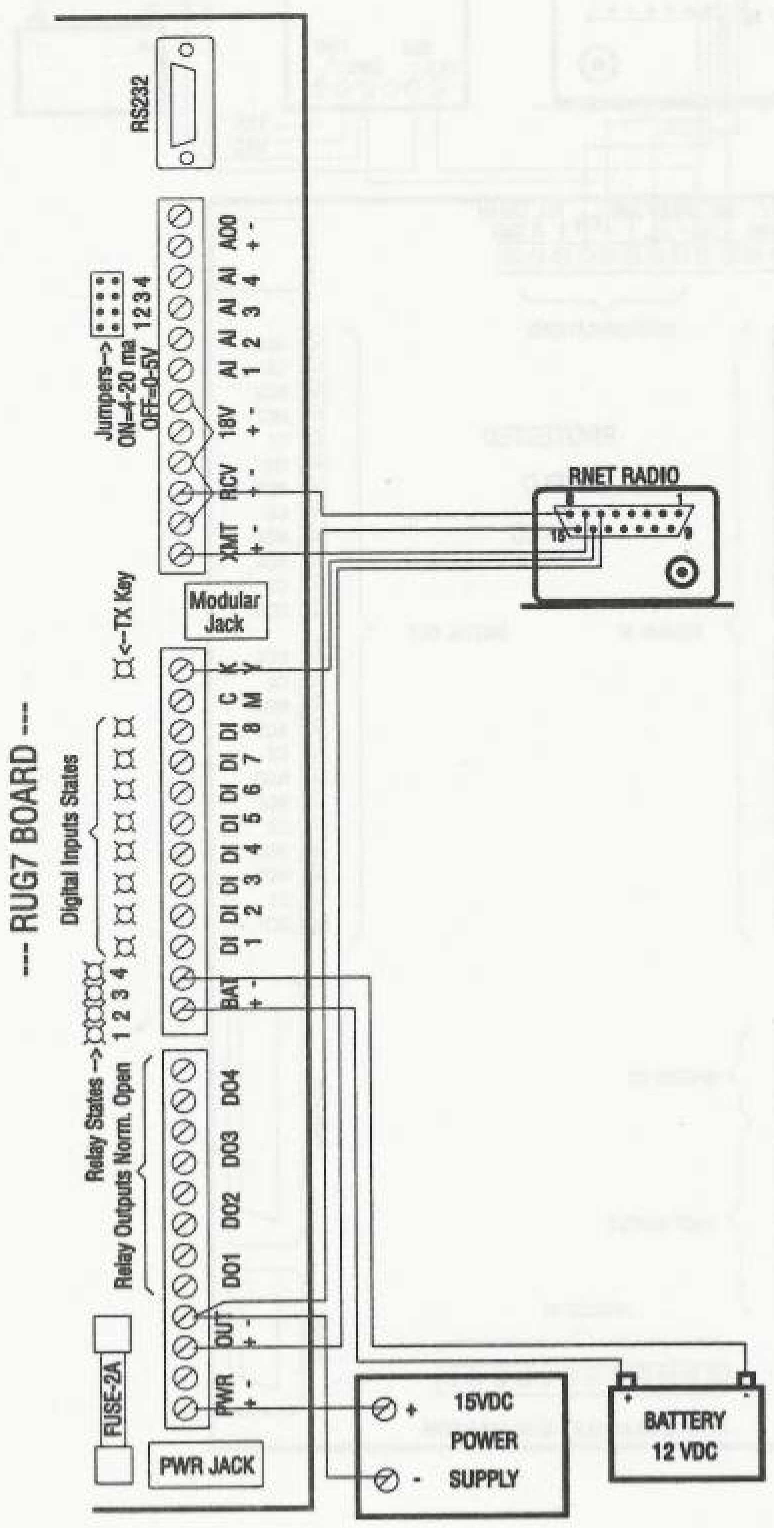


FIGURE 6.F.2 TYPICAL RUG7 RADIO CONNECTION

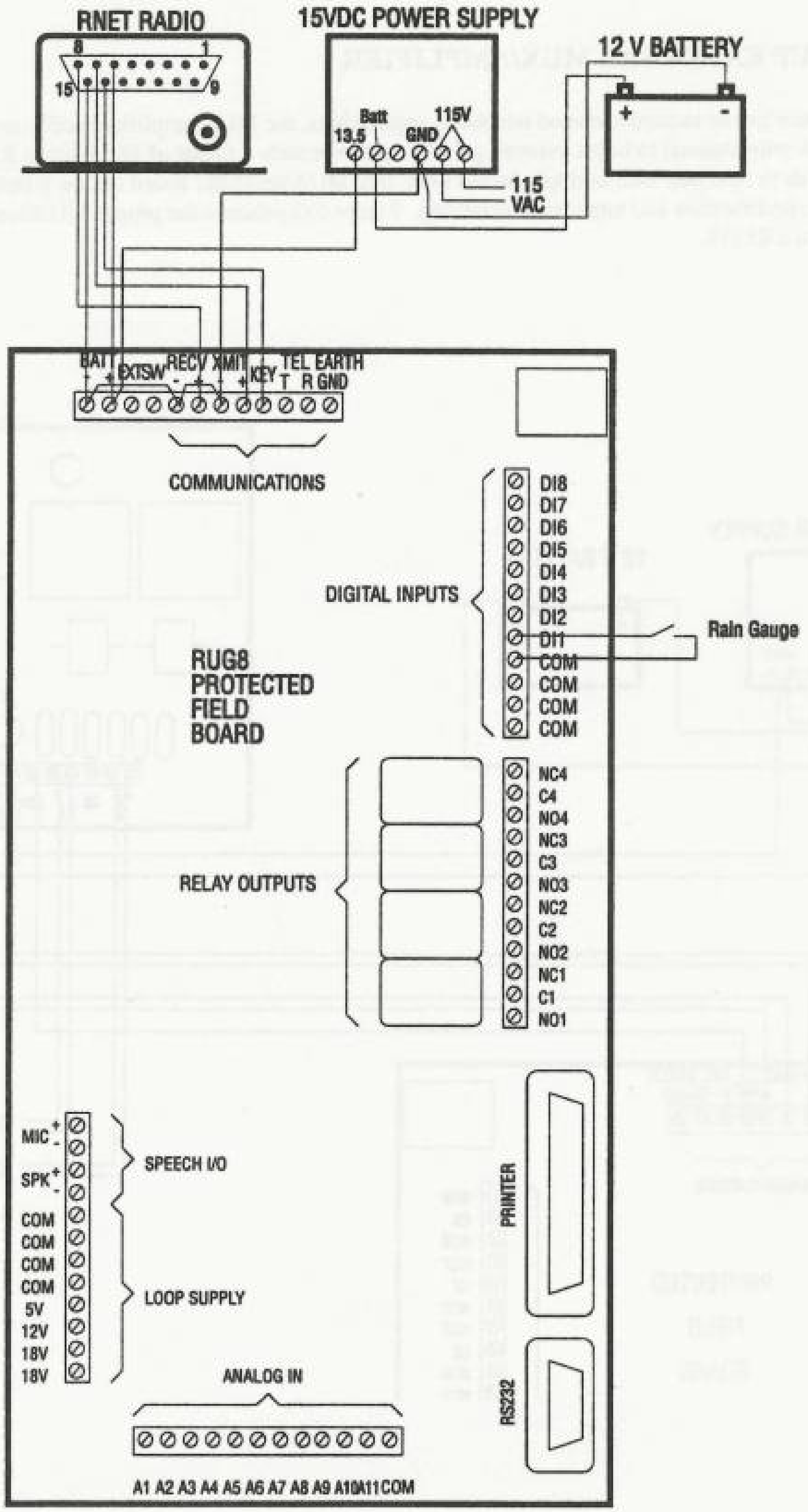


FIGURE 6.F.3 TYPICAL RUG8 RADIO CONNECTION

6.G HOOKUP EXTERNAL MUX/AMPLIFIER

For lease line or customer owned telephone applications, the MUX/amplifier board can be attached to the RUGID 4-wire channel to boost transmit gain by approximately a factor of 10. When a RUGID master unit needs to interface with multiple leased lines, one MUX/amplifier board can be installed for each line to provide amplification and impedance matching. Figure 6.G presents the proper MUX/amplifier board hookup to a RUG6.

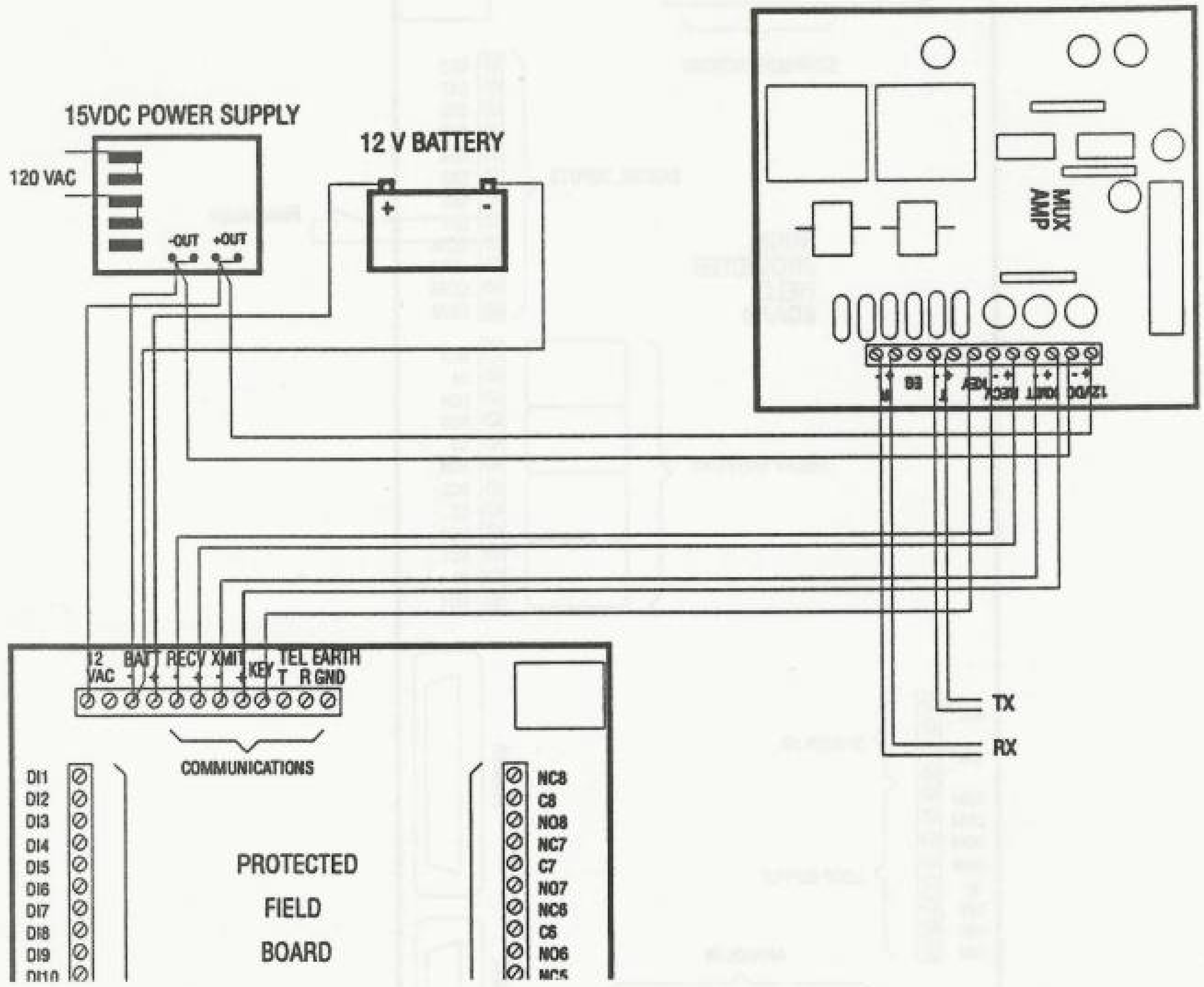


FIGURE 6.G MUX/AMP HOOKUP TO RUG6

SECTION 7

7.0 TROUBLESHOOTING

To troubleshoot a RUGID computer to the component level requires test equipment and knowledge of the design that are beyond the capability of most users. Outright failures should be referred to the manufacturer for diagnosis and repair. However, there are some conditions that may be user correctable. These are identified in this section.

7.1 PROGRAM IS LOST ON POWER OUTAGE

Normally, the onboard Lithium battery will provide power to the memory and real time clock during outages. Proceed as follows:

1. Enter a small program in memory and start it running. Note the time on the real time clock.
2. Remove power for 30 seconds then reapply it.
3. If the program is lost and the real time clock time is wrong, an excessive current drain is occurring in the battery load or the battery is dead or missing. Return the unit to the manufacturer for repair.
4. If the program is intact, remove power for one hour then reapply it.
5. If the program is lost, replace the Lithium battery. If that does not cure the problem, return the unit to the manufacturer for repair.

7.2 ANALOG INPUTS ARE INACCURATE

Inaccurate analog measurements are often caused by noise in the wiring bringing in the analog measurement. Observing the following practices will eliminate most noise problems:

1. Use shielded twisted pair wires.
2. Ground the shield only at the RUGID computer COM terminal.
3. Use sufficiently large wire to minimize voltage drop.
4. Do not run analog wiring parallel to wires carrying AC or switched signals.
5. Use large enough power supplies to handle worst case current and voltage requirements.

Make sure that during calibration the values applied to the transducer do not exceed the transducer's linear range, and that they do not exceed the linear range of any signal converters or other instruments using the same measurement. Evidence of a nonlinearity during calibration is accurate calibration near one end of the active range and increasing error as the other end is approached.

Ground loops are an additional source of error. Make sure that current flowing to or from another device does not have a path to flow through an analog signal wire. Note that the RUGID main board analog inputs have common analog returns. In other words, they are not isolated from each other. They must therefore be the last measurements in current loops before the loops are returned to the power supply negative terminal if more than one loop is used.

7.3 UNIT BOOTS TO MONITOR INSTEAD OF BASIC PROGRAM

Failure of the unit to boot to the same mode that existed before a power outage is due either to low battery voltage supplying power to the RAM that contains the reset vectors and mode specifier, or the executing program is writing erroneous data into the vector area. Check that your program is not writing to the system areas designated as reserved in the memory map, Appendix D. This can also be caused by the unit being hit by a transient due to lightning or back EMF from a starter or solenoid. Replacing the onboard Lithium battery will correct low voltage to the RAM. Applying snubbers to inductive loads such as starters and solenoids will attenuate back EMF transients from those devices. If these actions do not correct the problem, return the unit to the factory for repair.

7.4 CLOCK WILL NOT KEEP ACCURATE TIME

The real time clock uses a 32 Khz crystal as its time base. It should provide .005% rate accuracy. If the rate appears to have greater drift, return the unit to the factory for repair.

7.5 UNIT APPEARS TOTALLY INOPERATIVE

This can be caused by a failure in the unit or a blown fuse. If replacing the fuse does not bring the unit into operation, return it to the factory for repair. Be sure to return the wall transformer if it has one.

This symptom can also occur if the serial port setup parameters do not match those of the terminal or computer you are using to communicate with the unit, and the unit has no display to indicate activity. If it is possible that the baud rate, parity selection, or other serial port selection could be different from that of your terminal or computer, you can cause RUGID to install the default port setup parameters of 9600 baud (port 1), 8 bit word, 1 stop bit, and disabled parity by momentarily grounding the non-maskable interrupt pin, pin 6 on the microprocessor. Port 2 will initialize to 300 baud. To do this, you must power the main board up and momentarily connect pin 6 to pin 1 on integrated circuit R65C02P2. On RUG6 and RUG8 units, this can be done without removing the CPU board (the one nearest the front flanged plate, often with the LCD mounted on it). Instead, remove the steel case top cover and locate the white circle on the circuit board next to the power connection, a brown two pin connector. In the center of the white circle are two tinned feed through pads. With power applied to the unit, momentarily short the two pads together. This will install the default baud rate, etc.

7.6 DISPLAY IS HARD TO READ

The LCD display has a limited viewing angle and contrast ratio due to the multiplexing technique used in accessing the liquid crystal material. Contrast adjustment is accomplished by adjusting the contents of location $0216 (534). If your unit has a keyboard, you can adjust the contrast by simultaneously holding down the decimal point and "8" keys or the decimal point and "9" keys. There will be several seconds delay between the adjustment and a change in the display due to low pass filtering. On RUG7's, the display is temperature compensated using an onboard temperature transducer. To re-initialize the compensation, clear the RUG7 unit type designator bits in the voted memory (Appendix D), install DIAGNOS7.CMP and run it. This will restore the temperature compensation. If this doesn't work, return the unit for repair.

SECTION 8

<u>8.0 CIRCUIT BOARDS</u>

8.1.1 RUG6 and 8 MAIN BOARD COMPONENT ARRANGEMENT

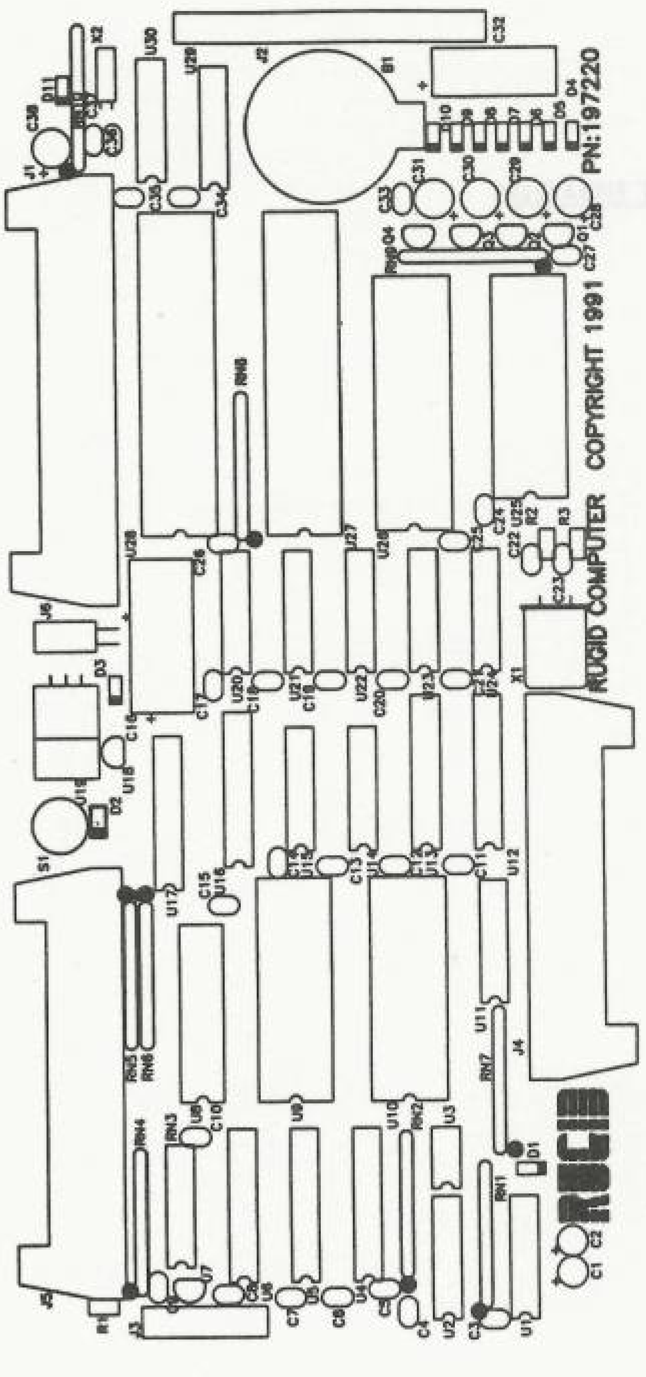


FIGURE 8.1 MAIN BOARD COMPONENT ARRANGEMENT

8.1.2 MAIN BOARD BUS CONNECTOR PINOUT

8.1.1.1 J1 Main Bus (Same for all boards)

PIN	FUNCTION	PIN	FUNCTION
1	877X speech enable	2	GND
3	IOEN, I/O enable	4	GND
5	A1	6	GND
7	No connection	8	GND
9	A2	10	GND
11	No connection	12	+5.7 VDC
13	A6	14	-10 VDC
15	A0	16	A5
17	A10	18	A8
19	+5VDC	20	A9
21	+5VDC	22	Unreg. in
23	+5VDC	24	Unreg. in
25	A7	26	Unreg. in
27	IRQ	28	A4
29	A3	30	R/W
31	CK2, 1.8 Mhz	32	RST
33	D7	34	D6
35	D5	36	D4
37	D3	38	D2
39	D1	40	D0

8.1.1.2 J4 PRINTER, RS232, RS422

PIN	FUNCTION	PIN	FUNCTION
1	Strobe invert	2	No connection
3	D1	4	Error from printer
5	D2	6	No connection
7	D3	8	No connection
9	D4	10	GND
11	D5	12	GND
13	D6	14	GND
15	D7	16	GND
17	D8	18	GND
19	ACK invert	20	GND
21	BUSY from printer	22	GND
23	PE from printer	24	GND
25	GND	26	RS232 transmit data
27	RS232 receive data	28	RS232 request to send
29	RS232 data set ready	30	No connection
31	No connection	32	RS232 carrier detect
33	RS422 +data	34	RS422 -data

8.1.1.3 J5 Main Board Field I/O Connection

MAIN BOARD J5 PIN NUMBER	FIELD BOARD PIN NUMBER	FUNCTION
1	T1-1	Analog input 1
2	T1-3	Analog input 2
3	T1-5	Analog input 3
4	T1-7	Analog input 4
5	T1-9	Analog input 5
6	T1-11	Analog input 6
7	T2-1	Analog input 7
8	T2-3	Analog input 8
9	T2-5	Analog input 9
10	T2-7	Analog input 10
11	T2-9	Analog input 11
12	T1-2,4,6,8,10,12	Analog common
	T2-2,4,6,8,10,12	Analog common
13	T3-9,10,11,12	GND
14	T4-9,10,11,12	GND
15		GND
16		Unregulated +12
17		Relay #1 driver
18		Relay #2 driver
19		Relay #3 driver
20		Relay #4 driver
21		Relay #5 driver
22		Relay #6 driver
23		Relay #7 driver
24		Relay #8 driver
25	T3-1 isolated	Digital input #1
26	T3-2 isolated	Digital input #2
27	T3-3 isolated	Digital input #3
28	T3-4 isolated	Digital input #4
29	T3-5 isolated	Digital input #5
30	T3-6 isolated	Digital input #6
31	T3-7 isolated	Digital input #7
32	T3-8 isolated	Digital input #8
33	T4-1 isolated	Digital input #9
34	T4-2 isolated	Digital input #10
35	T4-3 isolated	Digital input #11
36	T4-4 isolated	Digital input #12
37	T4-5 isolated	Digital input #13
38	T4-6 isolated	Digital input #14
39	T4-7 isolated	Digital input #15
40	T4-8 isolated	Digital input #16

8.2 RAM BANK COMPONENT ARRANGEMENT

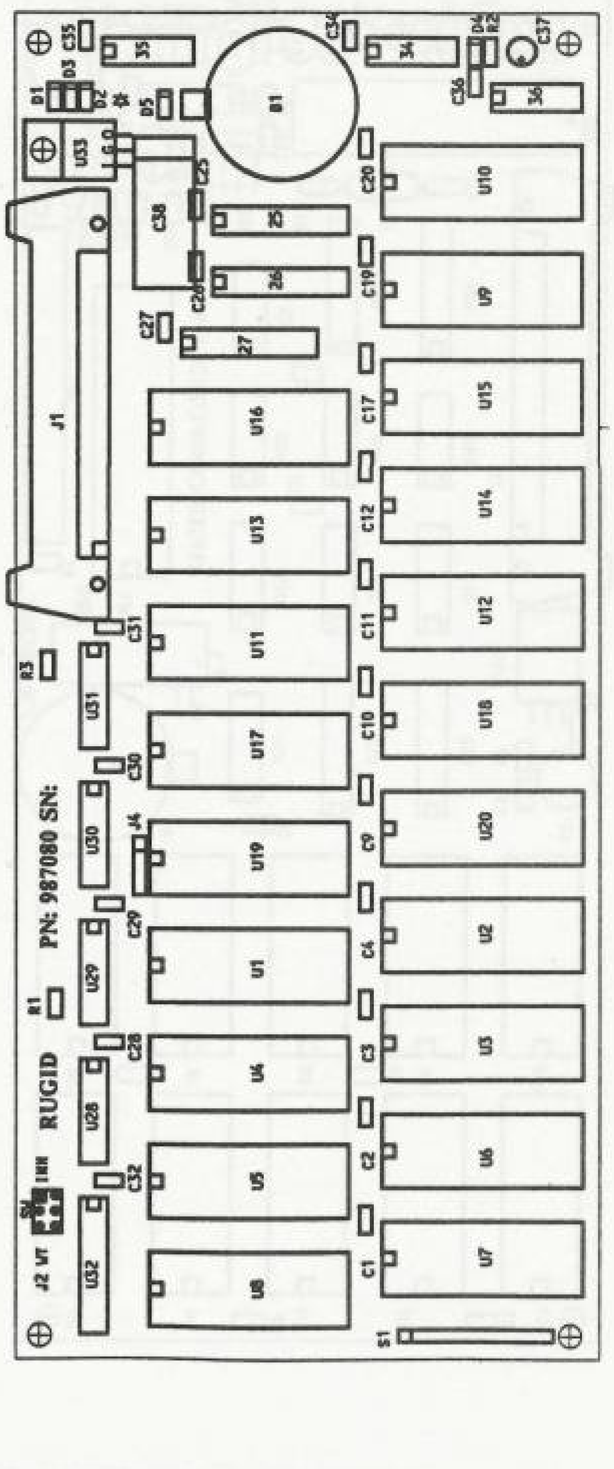


FIGURE 8.2 RAM BANK COMPONENT ARRANGEMENT

8.3 SPEECH SYNTHESIZER COMPONENT ARRANGEMENT

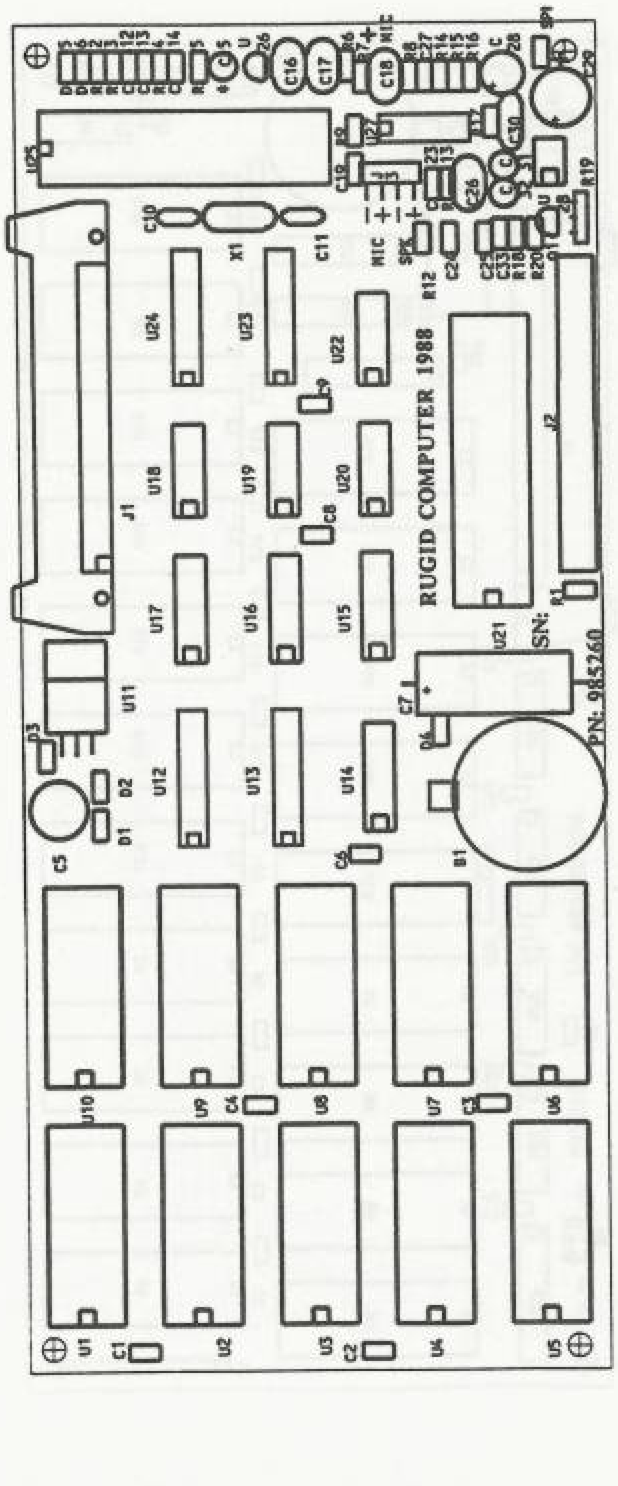


FIGURE 8.3 SPEECH SYNTHESIZER COMPONENT ARRANGEMENT

8.4 RUG6 MODEM BOARD

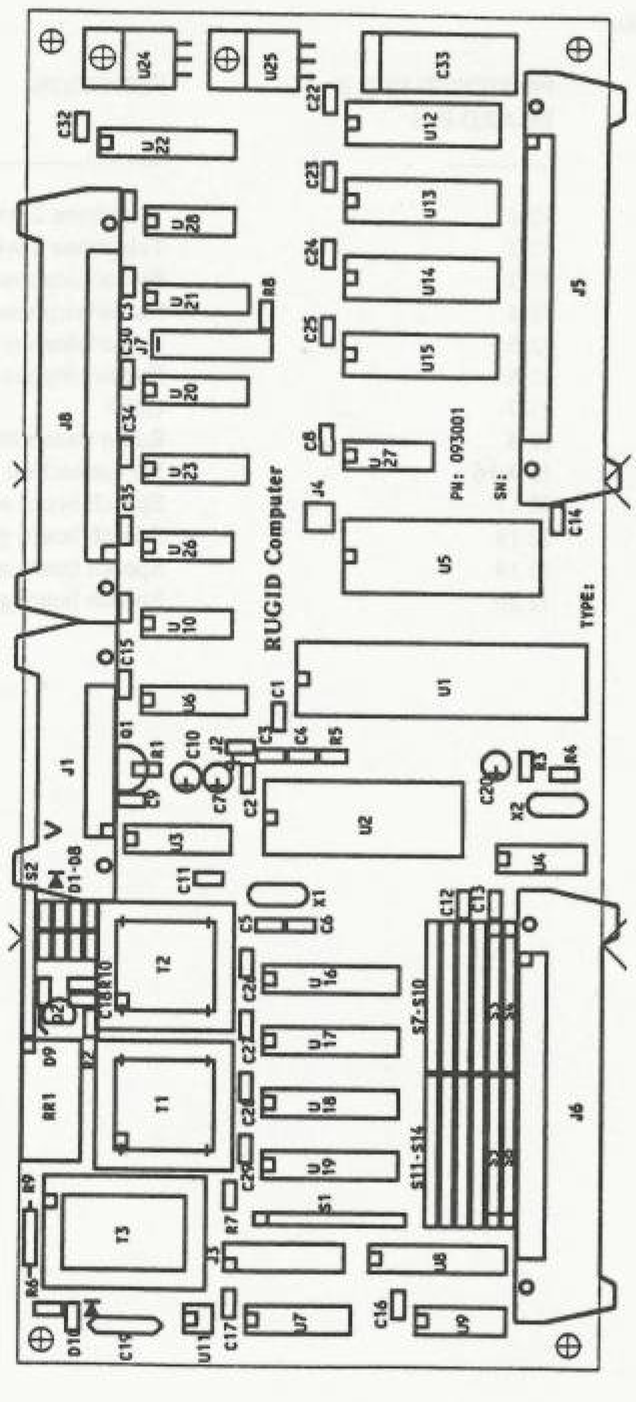


FIGURE 8.4.1 MODEM + DIGITAL I/O COMPONENT ARRANGEMENT

8.4.1 RUG6 MODEM + DIGITAL I/O CONNECTOR PINOUT

8.4.1.1 J1 Analog Communications Pinout

Applies to PN 093001

MODEM BRD. PIN NUMBER	PROTECED FIELD BOARD PIN	FUNCTION
1	J2-1	Telephone 2 wire +
2	J2-2	Telephone 2 wire -
3	J2-3	Radio/telephone 4 wire receive +
4	J2-4	Radio/telephone 4 wire receive -
5	J2-5	Radio/telephone 4 wire xmit +
6	J2-6	Radio/telephone 4 wire xmit -
7	J2-7	GND
8	J2-8	Radio transmitter keyer
9-16	J2 9-16	No connection
17	J2 17	Speech board mic input
18	J2 18	Speech board ground
19	J2 19	Speech board audio output
20	J2 20	Speech board ground

8.4.1.2 J5 Relay Driver Outputs

Applies to PN 093001

MODEM BRD. PIN NUMBER	EXPAN. FIELD BOARD PIN	FUNCTION
1	T1-1	Relay driver #1
2	T1-2	Relay driver #2
3	T1-3	Relay driver #3
4	T1-4	Relay driver #4
5	T1-5	Relay driver #5
6	T1-6	Relay driver #6
7	T1-7	Relay driver #7
8	T1-8	Relay driver #8
9	T1-9	Relay driver #9
10	T1-10	Relay driver #10
11	T1-11	Relay driver #11
12	T1-12	Relay driver #12
13	T2-1	Relay driver #13
14	T2-2	Relay driver #14
15	T2-3	Relay driver #15
16	T2-4	Relay driver #16
17	T2-5	Relay driver #17
18	T2-6	Relay driver #18
19	T2-7	Relay driver #19
20	T2-8	Relay driver #20
21	T2-9	Relay driver #21
22	T2-10	Relay driver #22
23	T2-11	Relay driver #23
24	T2-12	Relay driver #24
25	T3-1	Relay driver #25
26	T3-2	Relay driver #26
27	T3-3	Relay driver #27
28	T3-4	Relay driver #28
29	T3-5	Relay driver #29
30	T3-6	Relay driver #30
31	T3-7	Relay driver #31
32	T3-8	Relay driver #32
33	T3-9	Channel 1-8 EMF protect
34	T4-1	GND
35	T3-10	Channel 9-16 EMF protect
36	T4-2	GND
37	T3-11	Channel 17-24 EMF protect
38	T4-3	GND
39	T3-12	Channel 25-32 EMF protect
40	T4-4	GND

8.4.1.3 J6 Digital Input Expansion Pinout

Applies to PN 093001 only

MODEM BOARD PIN NUMBER	EXPAN. FIELD BOARD PIN	FUNCTION
J6-1	T1-1	Digital input #1
J6-2	T1-2	Digital input #2
J6-3	T1-3	Digital input #3
J6-4	T1-4	Digital input #4
J6-5	T1-5	Digital input #5
J6-6	T1-6	Digital input #6
J6-7	T1-7	Digital input #7
J6-8	T1-8	Digital input #8
J6-9	T1-9	Digital input #9
J6-10	T1-10	Digital input #10
J6-11	T1-11	Digital input #11
J6-12	T1-12	Digital input #12
J6-13	T2-1	Digital input #13
J6-14	T2-2	Digital input #14
J6-15	T2-3	Digital input #15
J6-16	T2-4	Digital input #16
J6-17	T2-5	Digital input #17
J6-18	T2-6	Digital input #18
J6-19	T2-7	Digital input #19
J6-20	T2-8	Digital input #20
J6-21	T2-9	Digital input #21
J6-22	T2-10	Digital input #22
J6-23	T2-11	Digital input #23
J6-24	T2-12	Digital input #24
J6-25	T3-1	Digital input #25
J6-26	T3-2	Digital input #26
J6-27	T3-3	Digital input #27
J6-28	T3-4	Digital input #28
J6-29	T3-5	Digital input #29
J6-30	T3-6	Digital input #30
J6-31	T3-7	Digital input #31
J6-32	T3-8	Digital input #32
J6-33	T3-9	+5VDC
J6-34	T4-1	GND
J6-35	T3-10	+5VDC
J6-36	T4-2	GND
J6-37	T3-11	Unreg. 12VDC
J6-38	T4-3	GND
J6-39	T4-4	GND

8.5 ANALOG I/O EXPANSION

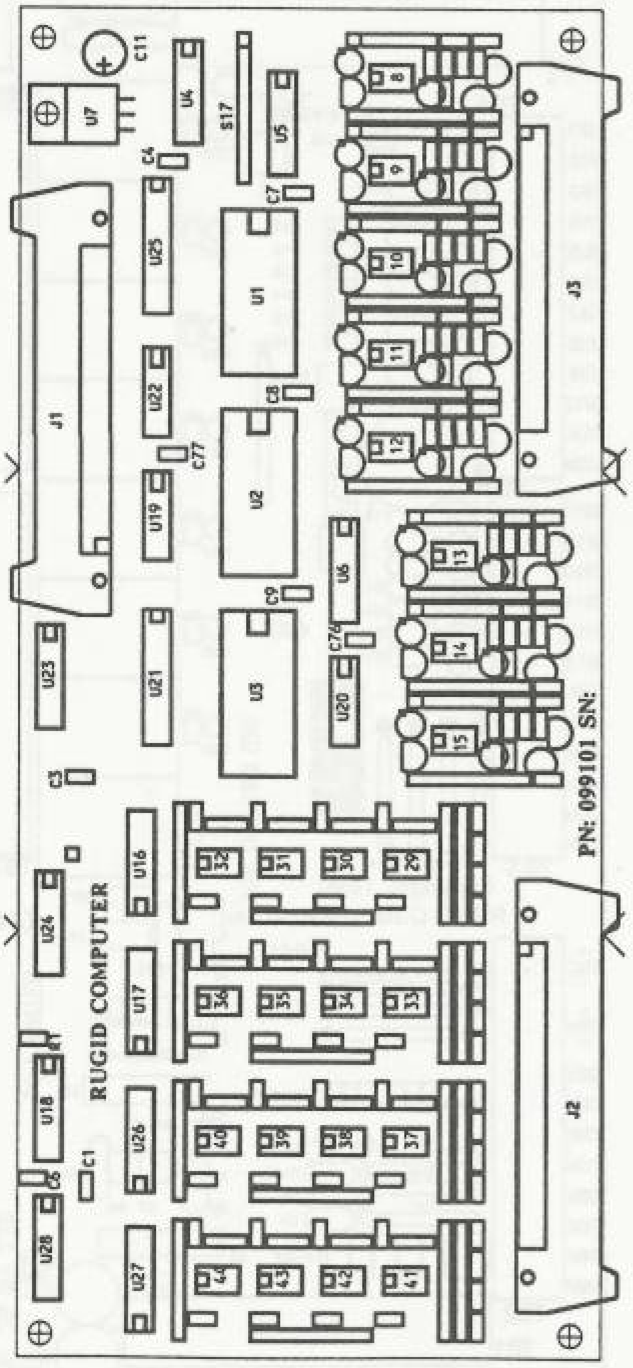


FIGURE 8.5 ANALOG I/O EXPANSION BOARD LAYOUT

8.6 EXTERNAL FIELD TERMINAL BOARDS

8.6.1 RUG6 PROTECTED FIELD BOARD

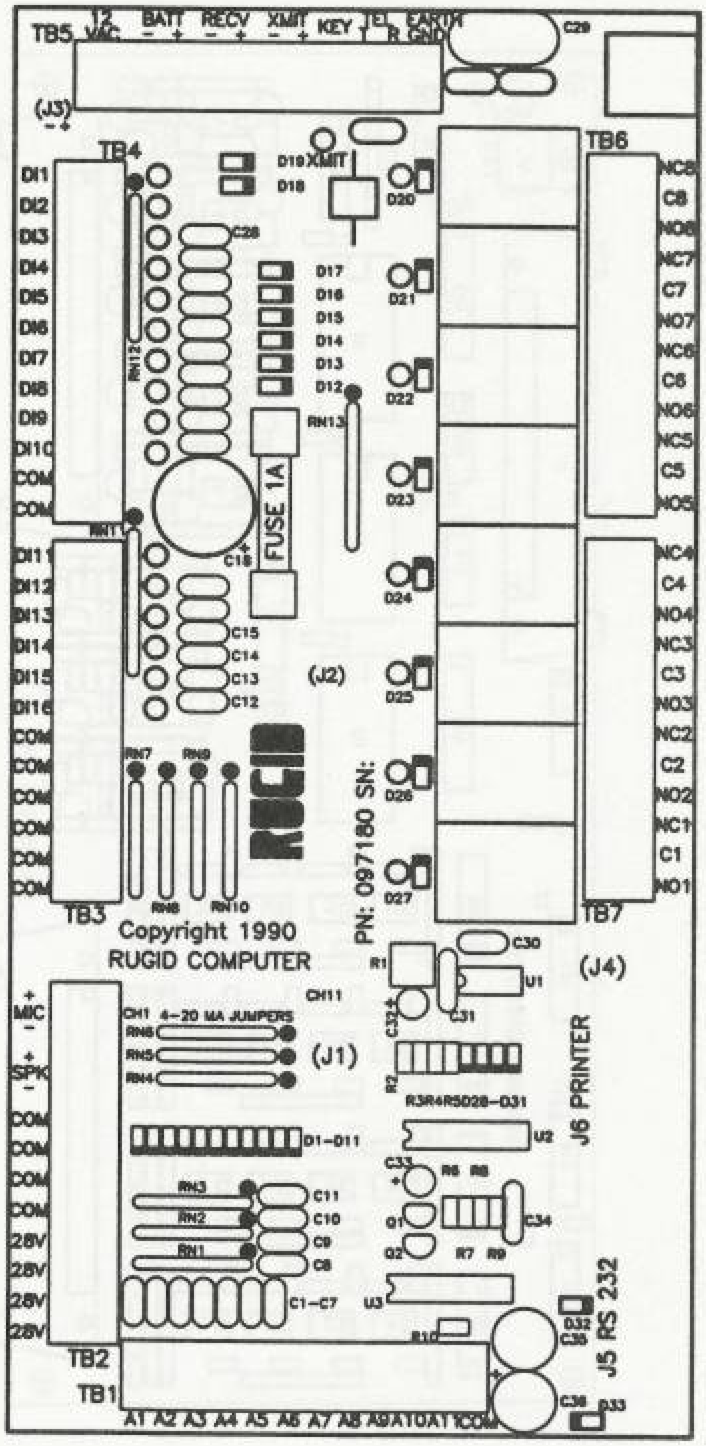


FIGURE 8.6.1 PROTECTED FIELD BOARD COMPONENT ARRANGEMENT

8.6.2 I/O EXPANSION TERMINAL BOARD (AFB, DFB)

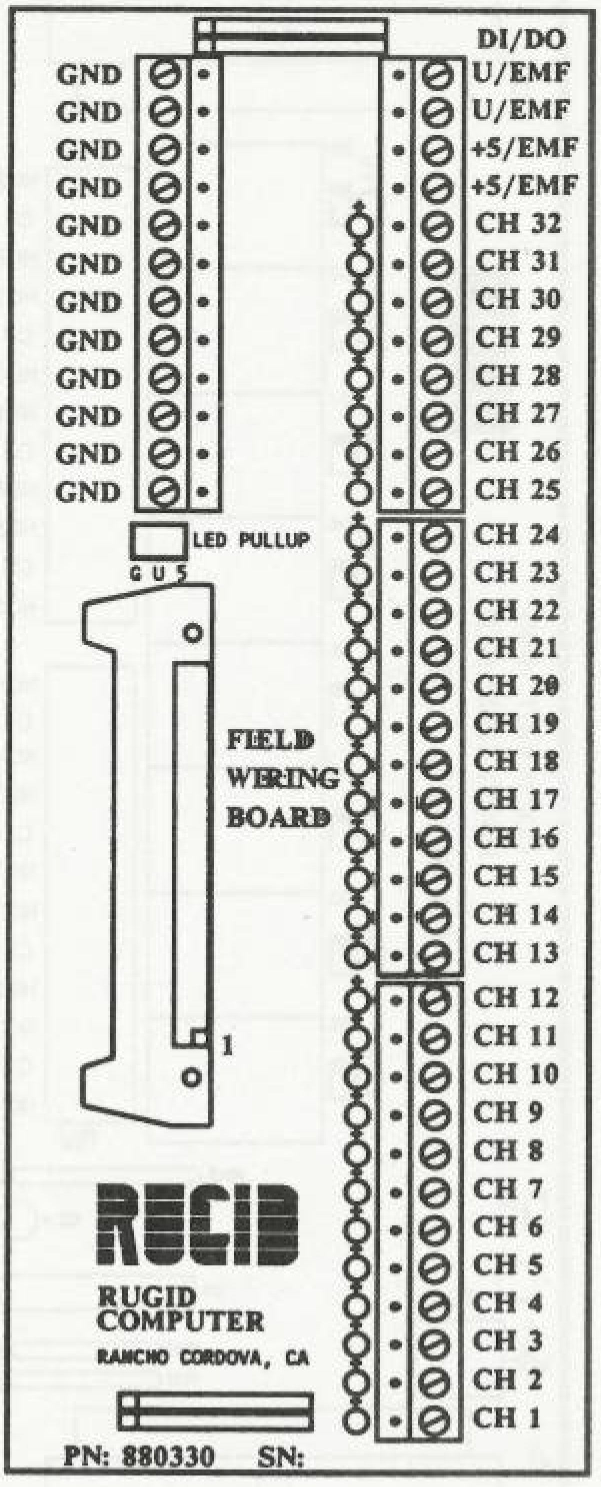


FIGURE 8.6.2 EXPANSION FIELD BOARD LAYOUT (AFB, DFB)

8.6.3 RELAY FIELD BOARD (RFB)

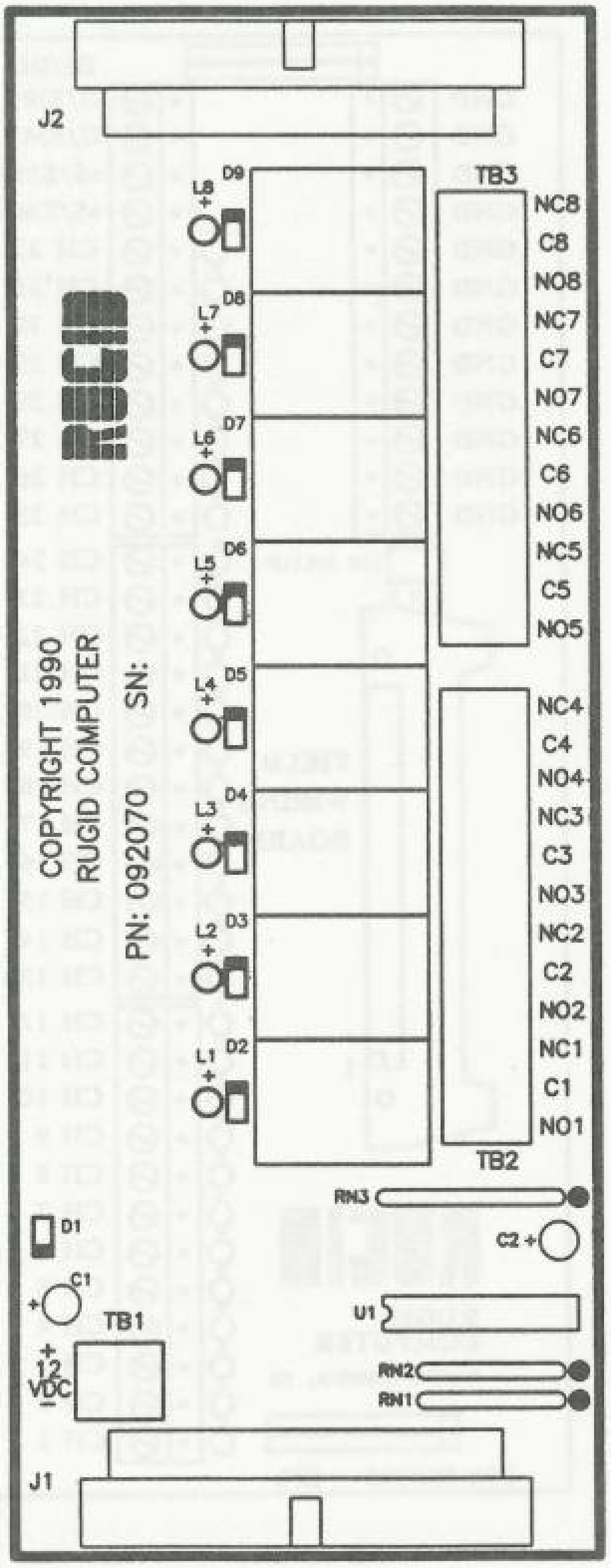


FIGURE 8.6.3 RELAY FIELD BOARD (RFB)

8.6.4 OPTICALLY ISOLATED FIELD BOARD (OFB)

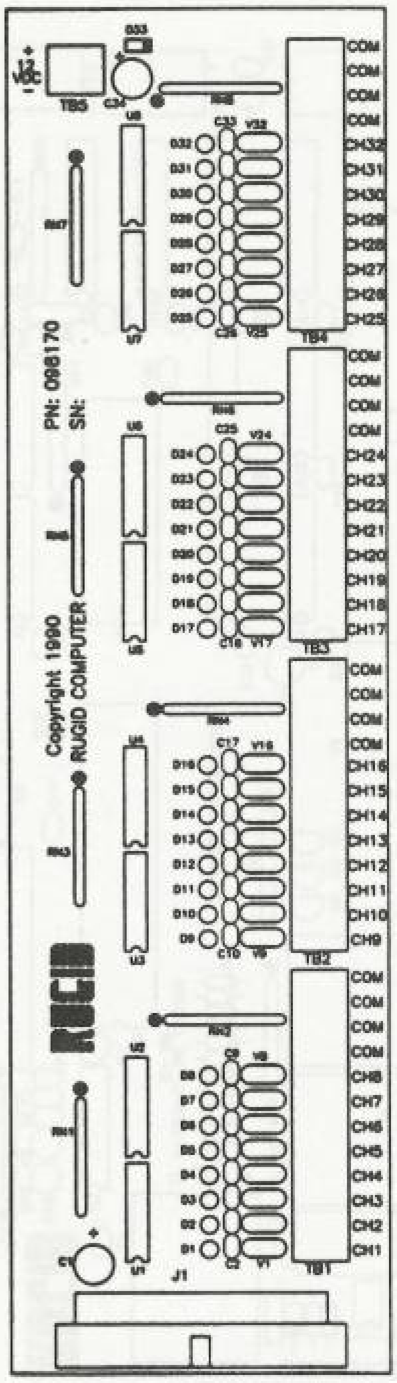


FIGURE 8.6.4 OPTICALLY ISOLATED FIELD BOARD LAYOUT

8.7 RUG8 BOARD ARRANGEMENTS

8.7.1 RUG8 COMMUNICATIONS BOARD COMPONENT ARRANGEMENT (COM)

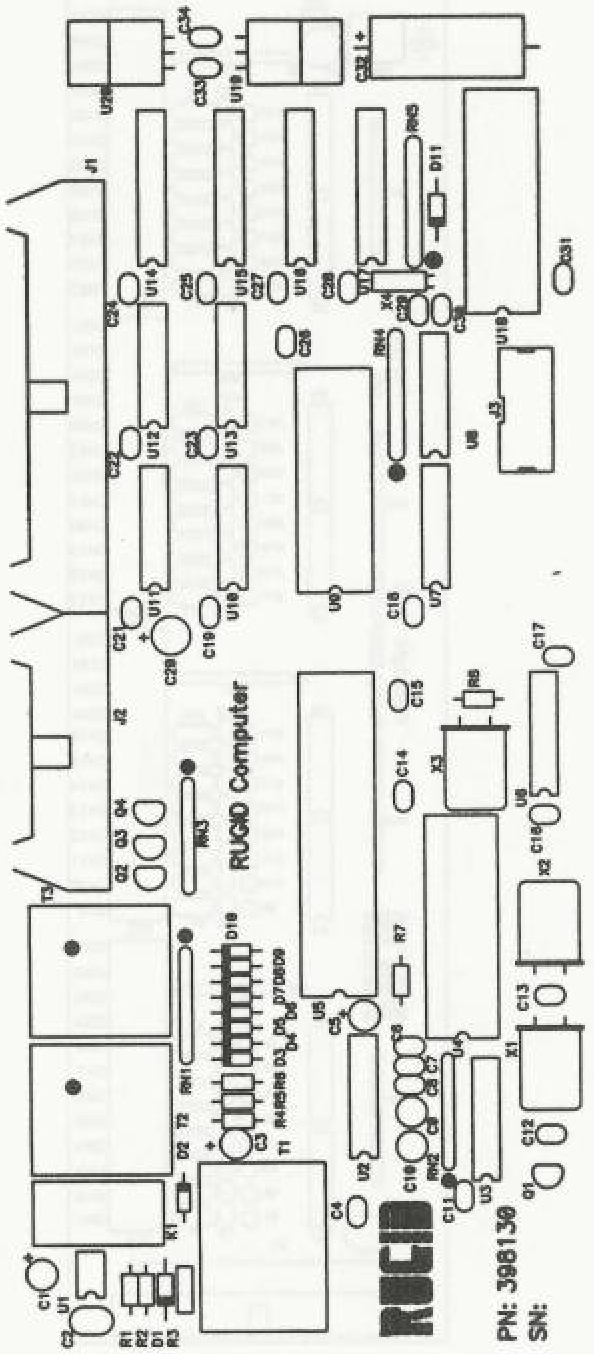


FIGURE 8.7.1 RUG8 COMMUNICATIONS BOARD COMPONENT ARRANGEMENT

8.7.2 RUG8 LOW POWER FIELD BOARD COMPONENT ARRANGEMENT (LFB)

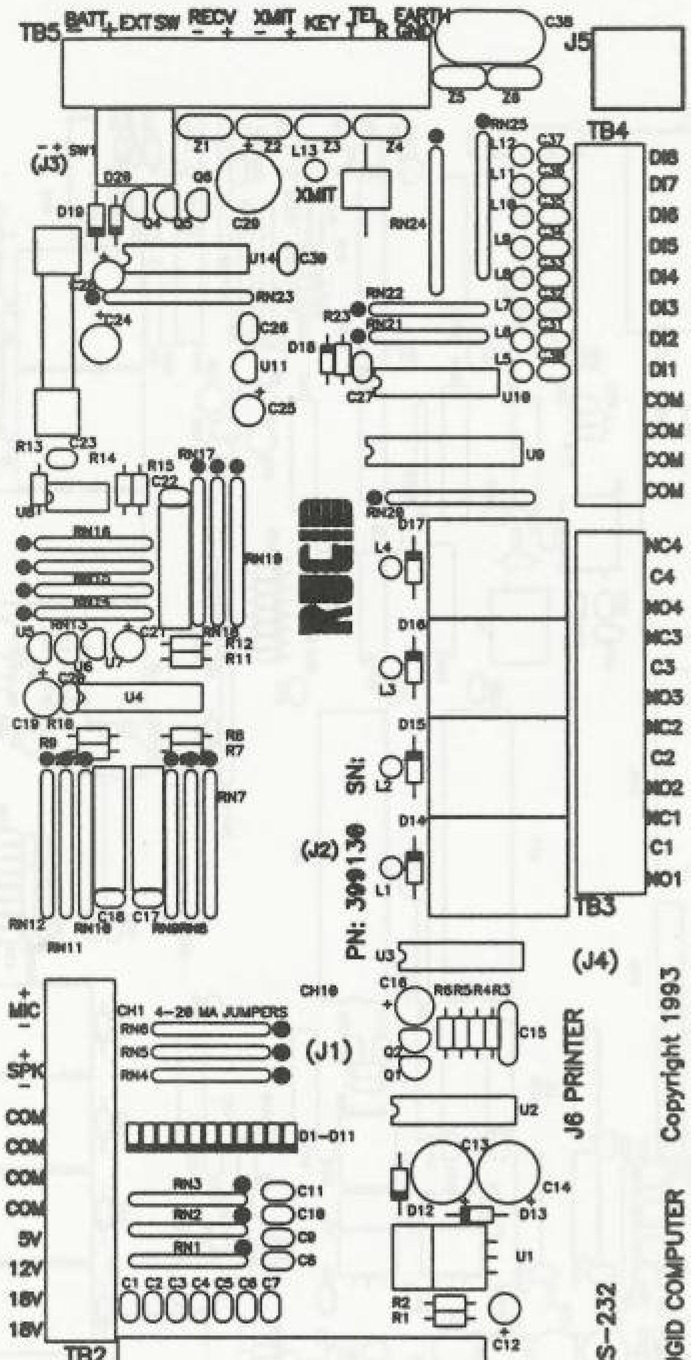


FIGURE 8.7.2 RUG8 LOW POWER FIELD BOARD

8.8 RUG7 BOARD COMPONENT ARRANGEMENT

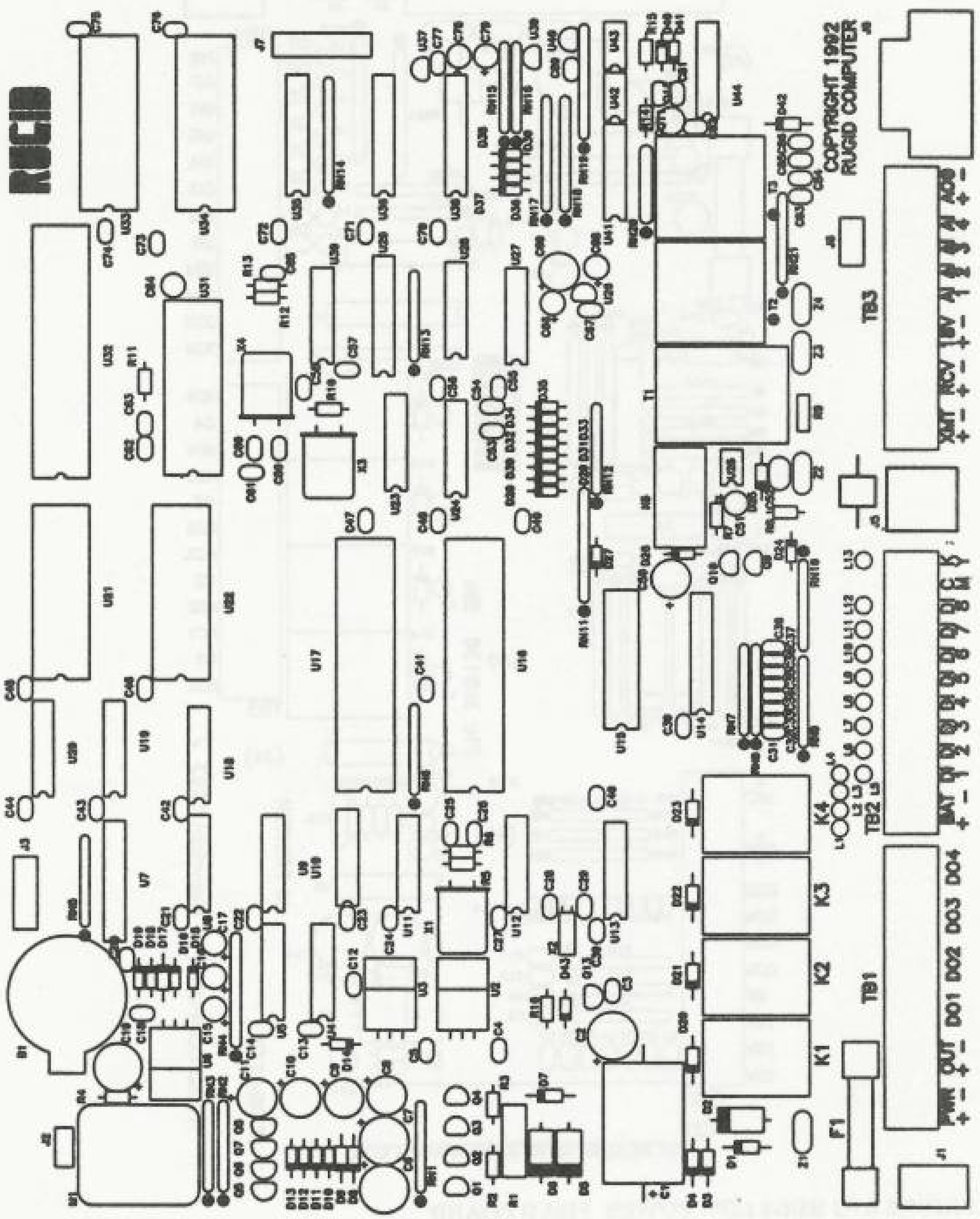


FIGURE 8.8 RUG7 BOARD COMPONENT ARRANGEMENT

8.9 MUX BOARD COMPONENT ARRANGEMENT

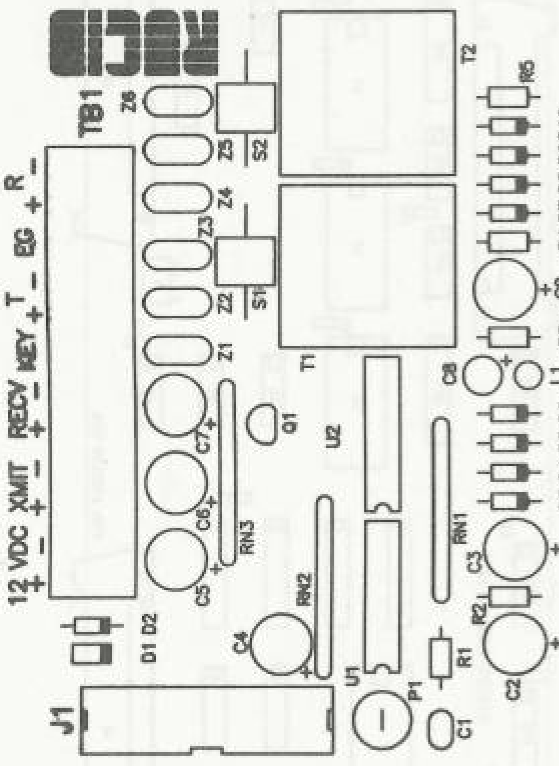


FIGURE 8.9 MUX BOARD COMPONENT ARRANGEMENT

8.A RADAR GUN/PROTOTYPING BOARD COMPONENT ARRANGEMENT

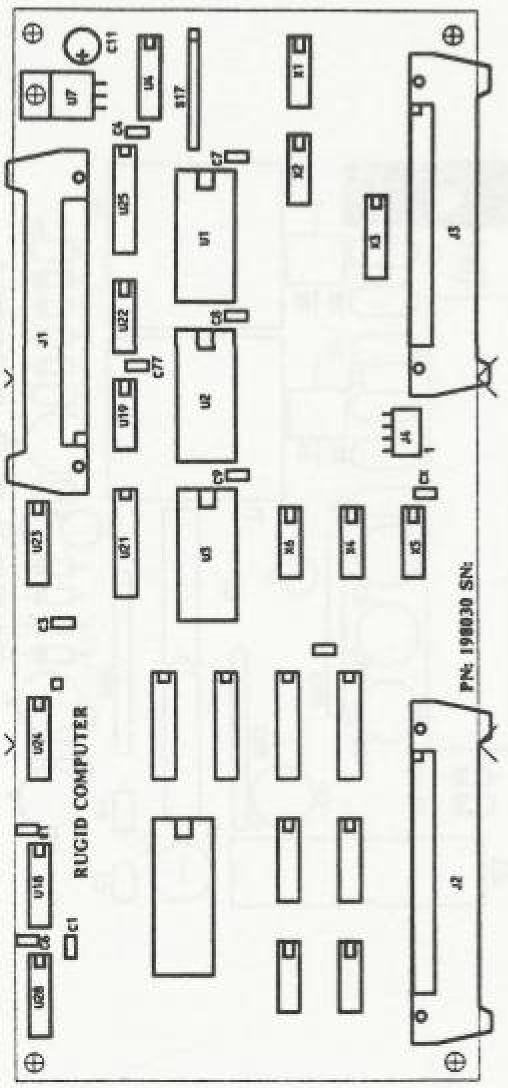


FIGURE 8.A RADAR GUN/PROTOTYPING BOARD COMPONENT ARRANGEMENT

SECTION 9

9.0 USEFUL INFORMATION

9.1 ASCII CHARACTER SET

Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex
----	---	---	----	---	---	----	---	---	----	---	---
NUL	0	0	SPC	32	20	@	64	40	?	96	60
SOH	1	1	!	33	21	A	65	41	a	97	61
STX	2	2	"	34	22	B	66	42	b	98	62
ETX	3	3	#	35	23	C	67	43	c	99	63
EOT	4	4	$	36	24	D	68	44	d	100	64
ENQ	5	5	%	37	25	E	69	45	e	101	65
ACK	6	6	&	38	26	F	70	46	f	102	66
BEL	7	7	'	39	27	G	71	47	g	103	67
BS	8	8	(	40	28	H	72	48	h	104	68
HT	9	9	)	41	29	I	73	49	i	105	69
LF	10	A	*	42	2A	J	74	4A	j	106	6A
VT	11	B	+	43	2B	K	75	4B	k	107	6B
FF	12	C	,	44	2C	L	76	4C	l	108	6C
CR	13	D	-	45	2D	M	77	4D	m	109	6D
SO	14	E	.	46	2E	N	78	4E	n	110	6E
SI	15	F	/	47	2F	O	79	4F	o	111	6F
DLE	16	10	0	48	30	P	80	50	p	112	70
DC1	17	11	1	49	31	Q	81	51	q	113	71
DC2	18	12	2	50	32	R	82	52	r	114	72
DC3	19	13	3	51	33	S	83	53	s	115	73
DC4	20	14	4	52	34	T	84	54	t	116	74
NAK	21	15	5	53	35	U	85	55	u	117	75
SYN	22	16	6	54	36	V	86	56	v	118	76
ETB	23	17	7	55	37	W	87	57	w	119	77
CAN	24	18	8	56	38	X	88	58	x	120	78
EM	25	19	9	57	39	Y	89	59	y	121	79
SUB	26	1A	:	58	3A	Z	90	5A	z	122	7A
ESC	27	1B	;	59	3B	[	91	5B	{	123	7B
FS	28	1C	<	60	3C	\	92	5C	|	124	7C
GS	29	1D	=	61	3D	]	93	5D	}	125	7D
RS	30	1E	>	62	3E	^	94	5E	->	126	7E
US	31	1F	?	63	3F	_	95	5F	<-	127	7F

LF = Line Feed   CR = Carriage Return   LF=^J   VT=^K
FF = Form Feed   ESC = Escape   FF=^L   CR=^M

Graphics mode only:

CHARACTER	DEC	HEX
Up arrow	128	$80
Down arrow	129	$81
|	138	$8A
||	139	$8B
|||	140	$8C
||||	141	$8D
|||||	142	$8E
||||||	143	$8F
|||||||	144	$90

9.2 ERROR MESSAGES

Next Without For

Each FOR instruction must be paired with a NEXT instruction which establishes the end of the instructions to be executed in the loop. You have probably forgotten to include the NEXT or have misspelled it.

Syntax

The interpreter has encountered an instruction with incorrect punctuation, an illegal character, a missing parentheses, etc.

Return Without GOSUB

A RETURN has been encountered before a GOSUB was executed. Make sure that your program has not "fallen through" into a subroutine.

Out of Data

A READ instruction has run out of data. Possibly a RESTORE instruction has been forgotten, or all the data to be read has not been accounted for.

Function Call Parameter

A missing or extra parameter exists in a function call; or one of the wrong type exists. This usually occurs as a result of one of the following:

1. Negative or large subscript
2. Negative or zero argument for LOG function
3. Negative argument for SQR function
4. A USR call before address establishment
5. An improper argument exists in a call to MID$, RIGHT$, LEFT$, WAIT, PEEK, POKE, ON...GOTO, SPC, or TAB.

Overflow

A mathematical expression has exceeded the interpreter's capacity to represent the result. This most commonly results from a divide by zero. When divisions are necessary, you must be certain to test in advance that the divisor is not zero.

Out of Memory

Attempting to add a statement or variable has resulted in exhaustion of the memory in either the volatile or nonvolatile memory area. If this error occurred when entering a program statement, nonvolatile RAM is full. If it occurred during execution, the first character of the variable being stored indicates the RAM area that is full. You will have to compress statements, reuse variables, or use integer arrays to reduce memory occupancy. This error can also occur if the stack overflows due to nesting FOR...NEXT or GOSUB statements too deeply, or nesting parenthetical expressions.

Undefined Statement

An attempt was made to GOTO, GOSUB or THEN to a statement number that does not exist.

Invalid Subscript

An attempt was made to access an array entry larger than was dimensioned, or the wrong number of subscripts was used. This often occurs when an array is referenced without being dimensioned.

Pointer Error

An array was dimensioned twice. This often occurs after BASIC dimensions an array automatically when referenced, and then encounters the DIM command for that array.

Attempted Divide by Zero

An attempt was made to divide by zero. You should check divisors before attempting division.

Illegal Direct Command

Certain commands such as DEF, GET, or INPUT cannot be executed in the command mode.

Data Type Mismatch

The left and right sides of an expression have a numeric/string mismatch, or a function is supplied with the wrong data type.

Long String

An attempt was made to create a string whose length exceeds 255 characters.

Complex String

A string expression is too complex. You must break the expression into two or more shorter expressions.

Continue Error

An attempt was made to continue a program when it has been altered, an error has occurred, or the program does not exist.

Undefined Function

An attempt was made to execute a user function that is not yet defined. Check the spelling of the function definition ar the call.

9.3 WARRANTY

RUGID COMPUTER WARRANTS THAT EQUIPMENT MANUFACTURED AND SOLD BY US IS FREE FROM DEFECTS IN MATERIAL AND WORKMANSHIP. UNDER THIS WARRANTY, OUR OBLIGATION IS LIMITED TO REPAIRING OR REPLACING, AT OUR OPTION, ANY EQUIPMENT OR PARTS RETURNED SHIPPING PREPAID AND PROPERLY PACKED TO OUR PLANT AND PROVING TO BE DEFECTIVE BY OUR INSPECTION WITHIN ONE YEAR AFTER SALE TO THE ORIGINAL PURCHASER. THIS WARRANTY SHALL NOT APPLY TO EQUIPMENT OR PARTS THEREOF WHICH ARE NORMALLY CONSUMED IN OPERATION, OR TO ANY EQUIPMENT WHICH SHALL HAVE BEEN REPAIRED OR ALTERED IN ANY WAY OUTSIDE OUR PLANT, SO AS TO, IN THE JUDGEMENT OF RUGID COMPUTER, AFFECT ITS STABILITY, ACCURACY, OR RELIABILITY, NOR WHICH HAS BEEN OPERATED IN A MANNER OR ENVIRONMENT EXCEEDING ITS SPECIFICATIONS, NOR WHICH HAS BEEN DAMAGED, ALTERED, DEFACED, OR HAS HAD ITS SERIAL NUMBER REMOVED. UNDER NO CIRCUMSTANCES SHALL RUGID COMPUTER BE LIABLE FOR ANY LOSS OR DAMAGE, DIRECT, INCIDENTAL OR CONSEQUENTIAL, ARISING OUT OF THE USE, MISUSE, OR INABILITY TO USE, THIS PRODUCT. THE LIABILITY OF RUGID COMPUTER SHALL NOT EXCEED THE ORIGINAL PURCHASE PRICE OF THIS PRODUCT.

9.4 RETURN/REPAIR POLICY

Specific warranty provisions are stated in the warranty section above. Under no circumstances are products returnable for credit. If a product is found to have failed, it must be returned to the factory for repair or replacement. The determination of whether to repair or replace the product is made by us after a period of testing. If it cannot be brought up to complete operation with full certainty, it will be replaced. When repaired under warranty, we will return the repaired product using the same freight class as it was received, no charge. We will make every effort to ship within 24 hours of receipt. The determination of whether a product's repair or replacement is covered under warranty will also be made by us. We give the benefit of any doubt to the customer in this determination. However, if there is any indication of physical damage, alteration, or misapplication of the product, then the repair will not be covered by the warranty.

It is in your best interest to accurately assess and report the circumstances of a failure. One of the most difficult determinations to make is whether a product has suffered overvoltage stress in the field. If it has, the product can fail a month after being repaired and declared operational due to failure of a component that was stressed but did not reveal itself during testing. Also, any failure related information you can give us along with the product can reduce the time it takes to find the defective components. Therefore, it could save you money.

9.5 MEMORY MAP

The following list presents the RUGID memory organization. Addresses are given in hexidecimal notation.

ADDRESS	FUNCTION
\$0000-0112	Scratchpad
\$0113-01FF	Stack
\$0200-0211	More scratchpad
\$0212-0250	Background software scratchpad
\$0251-02A3	Modem port output buffer
\$02A4-02F6	Terminal port output buffer
\$02F7-0349	Printer output buffer
\$034A-03D2	Background scratchpad
\$03D3-0425	Modem port receive buffer
\$0426-0479	Terminal port receive buffer
\$047A-04A0	Scratchpad
\$04A1-04C2	Pointers to preassigned arrays
\$04C3-04DA	Scratchpad
\$04DB-1EFF	BASIC variable storage area
\$1F00-1FFF	Reserved area for user data CAL.LIB uses this area to store calibration constants.
\$2000-7EFF	BASIC program
\$7F00-7F2F	Voted critical data
\$7F30-7F7F	Reserved system area
\$7F80-7FFF	Reserved area for user
\$8000-80FF	RAM bank 256 byte window
\$8100-81FF	RAM bank select addresses
\$8200-82FF	Analog I/O select addresses
\$8300-85FF	Reserved I/O select addresses
\$8600-86FF	Digital I/O select addresses
\$8700-87FF	Main board I/O select addresses (\$877X=Speech processor)

9.6 VOTED MEMORY CONTENTS

Memory locations $7F00 through 7F0F contain critical parameters that govern unit operation. The assignments of these locations are presented below along with the default values installed when the unit is newly initialized. The contents of these locations are replicated two more times in locations $7F10 through $7F1F and $7F20 through $7F2F for storage redundancy. Upon boot up, the unit compares the three copies of this data and, if a single byte mismatch is discovered, will make the mismatching byte match that of the other two. If more than one parameter is found to be in error, the unit will install the defaults identified below. These defaults will also be installed in the event of a non maskable interrupt, which occurs when pins 1 and 6 on the mocroprocessor (G65C02P2) are shorted together.

Address=$7F00 Default=$22 Purpose=mode setting: $22=Monitor, $43=BASIC, $8C=Command

Address=$7F01 Default=$01 Purpose=Modem UART command:

BYTE	7	6	5	4	3	2	1	0
FUNCT	PARITY: X X 0 No parity 0 0 1 Odd 0 1 1 Even 1 0 1 TX mark 1 1 1 TX space			ECHO: 0=OK 1=Echo	TX CONTROL: 0 0 IRQ off, RTS on, TX off 0 1 IRQ on, RTS off, TX on 1 0 IRQ off, RTS off, TX on 1 1 IRQ off, RTS off, TX bk		RCV IRQ: 0=on 1=off	DTR 0=no rcv 1=rcv ok

Address=$7F02 Default=$18 Purpose=Modem UART control:

BYTE	7	6	5	4	3	2	1	0
FUNCT	STOP BITS: 0=1 1=2	WORD LENGTH: 0 0 = 8 0 1 = 7 1 0 = 6 1 1 = 5		RCV CLK: 0=Ext 1=Int	BAUD RATE: 0 0 0 0 = Ext. 0 0 0 1 = 50 0 0 1 0 = 75 0 0 1 1 = 110 0 1 0 0 = 135 0 1 0 1 = 150 0 1 1 0 = 300 0 1 1 1 = 600		1 0 0 0 = 1200 1 0 0 1 = 1800 1 0 1 0 = 2400 1 0 1 1 = 3600 1 1 0 0 = 4800 1 1 0 1 = 7200 1 1 1 0 = 9600 1 1 1 1 = 19,200	

Address=$7F03 Default=$00 Purpose=Unit address for CRC communications

Address=$7F04 Default=$01 Purpose=Terminal UART command, see $7F01

BYTE	7	6	5	4	3	2	1	0
FUNCT	PARITY: X X 0 No parity 0 0 1 Odd 0 1 1 Even 1 0 1 TX mark 1 1 1 TX space			ECHO: 0=OK 1=Echo	TX CONTROL: 0 0 IRQ off, RTS on, TX off 0 1 IRQ on, RTS off, TX on 1 0 IRQ off, RTS off, TX on 1 1 IRQ off, RTS off, TX bk		RCV IRQ: 0=on 1=off	DTR 0=no rcv 1=rcv ok

Address=$7F05 Default=$1E Purpose=Terminal UART control, see $7F02

BYTE	7	6	5	4	3	2	1	0
FUNCT	STOP BITS: 0=1 1=2	WORD LENGTH: 0 0 = 8 0 1 = 7 1 0 = 6 1 1 = 5		RCV CLK: 0=Ext 1=Int	BAUD RATE: 0 0 0 0 = Ext. 0 0 0 1 = 50 0 0 1 0 = 75 0 0 1 1 = 110 0 1 0 0 = 135 0 1 0 1 = 150 0 1 1 0 = 300 0 1 1 1 = 600		1 0 0 0 = 1200 1 0 0 1 = 1800 1 0 1 0 = 2400 1 0 1 1 = 3600 1 1 0 0 = 4800 1 1 0 1 = 7200 1 1 1 0 = 9600 1 1 1 1 = 19,200	

Address=$7F06,7Default=set by EPROM Purpose=IRQ interrupt vector

Address=$7F08,9Default=set by EPROM Purpose=Reset vector

Address=$7F0A Default=$03 Purpose=Rings to answer, tone use:

BYTE	7	6	5	4	3	2	1	0
FUNCT	212/103/202: 0=103 1=202	TONE USE: 0 0 0 = Answer 0 0 1 = Originate 0 1 0 = High tones 0 1 1 =Low tones 1 0 0 = Autoset			RINGS TO ANSWER: Range=0 to 15 Sets rings to answer by background SW			

Address=$7F0B Default=$00 Purpose=CRC communications setup:

BYTE	7	6	5	4	3	2	1	0
FUNCT	DIAG- NOST: 0=off 1=on	Spare	DIAGNOSTIC PORT 0 0 = LCD 0 1 = Terminal 1 0 = Modem 1 1 = PRN		2WIRE/ 4WIRE 0=2W 1=4W	Spare	MODEM TLM MODE: 0 0 = RUGID 0 1 = CRC 1 0 = ASCII	

Address=$7F0C Default=$08 Purpose=Modem delays:

BYTE	7	6	5	4	3	2	1	0
FUNCT	# Nulls		Transmit delay in 56ths of a second					

Address=$7F0D Default=$00 Purpose=Unit type:

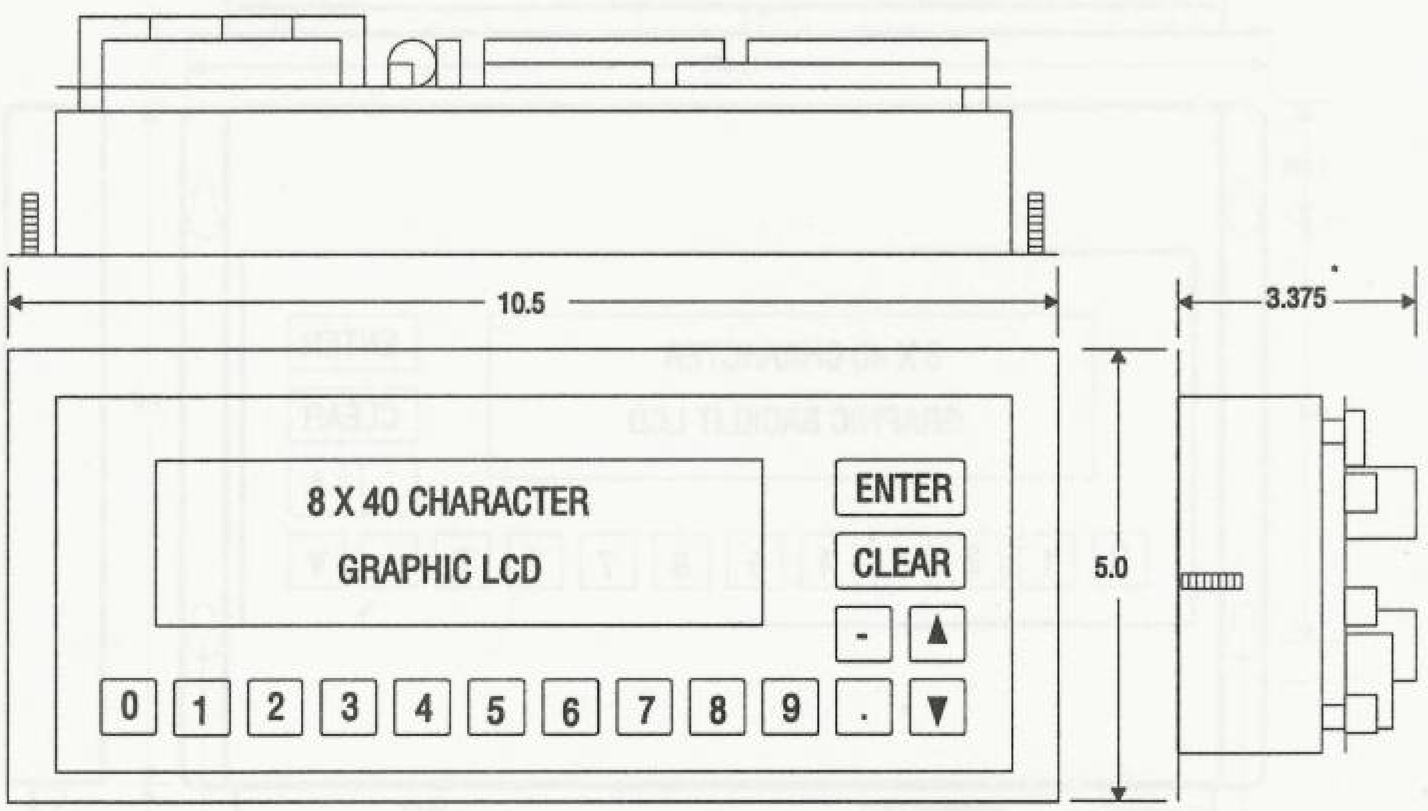
BYTE	7	6	5	4	3	2	1	0
FUNCT							UNIT TYPE: 0 0 = RUG6 0 1 = RUG7 1 0 = RUG8	

Address=$7F0E Default=$00 Purpose=Spare

Address=$7F0F Default=$20 Purpose=Misc. flags:

BYTE	7	6	5	4	3	2	1	0
FUNCT		CD 4053 control on boot	HC153 control on boot up		Enable Modbus on RAM bank 0=std 1=RAM bank		Port 1 mode 0=ASCII 1=MODBU S	Printer wait 0=for-ever 1=2.5 sec

APPENDIX A UNIT DIMENSIONS



NOTE: All dimensions in inches

* Depth of 5-slot case is 5.75 inches
Depth of 8-slot case is 8.0625 inches

Figure A.1 RUG6/8 Unit Dimensions

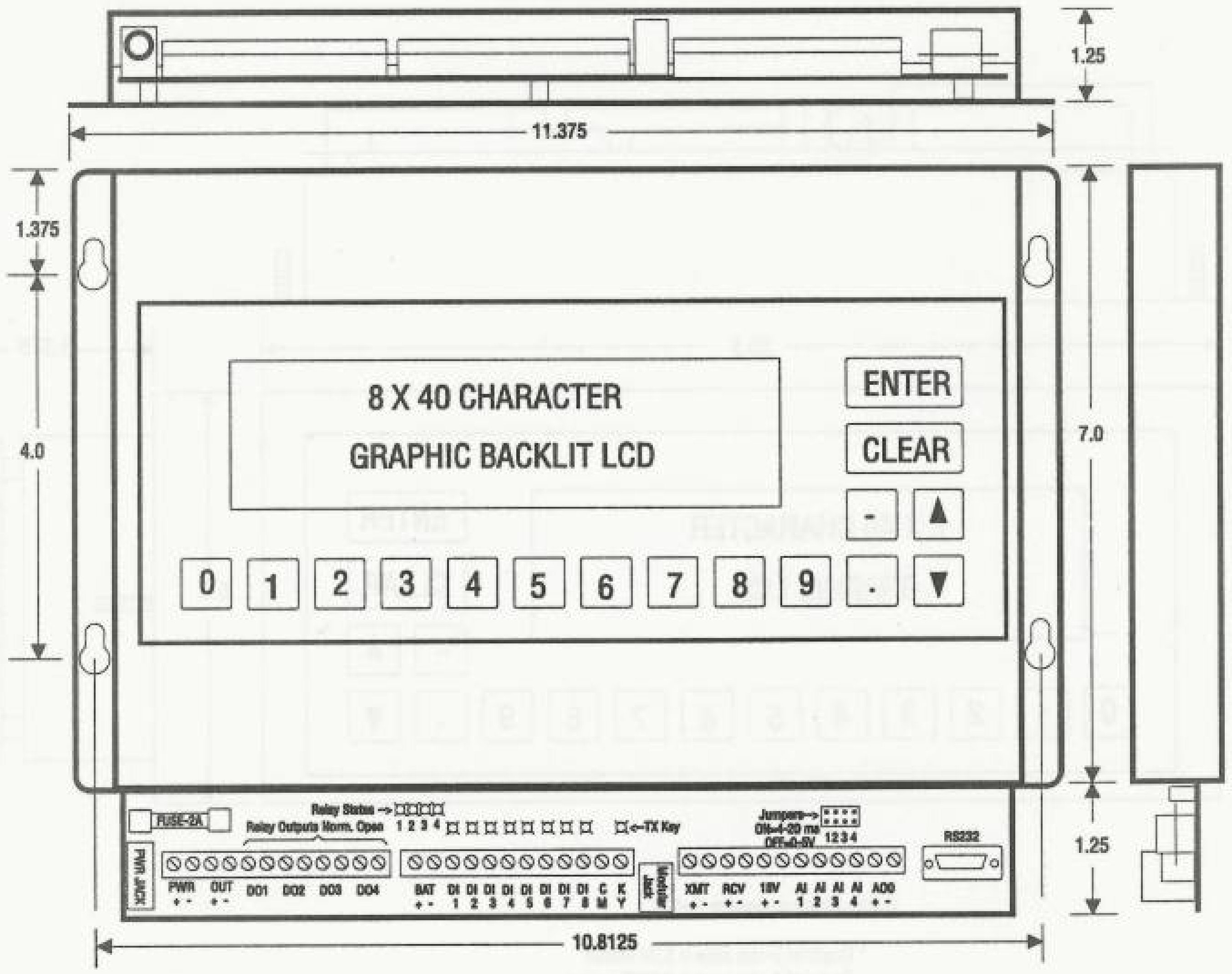


Figure A.2 RUG7 Unit Dimensions

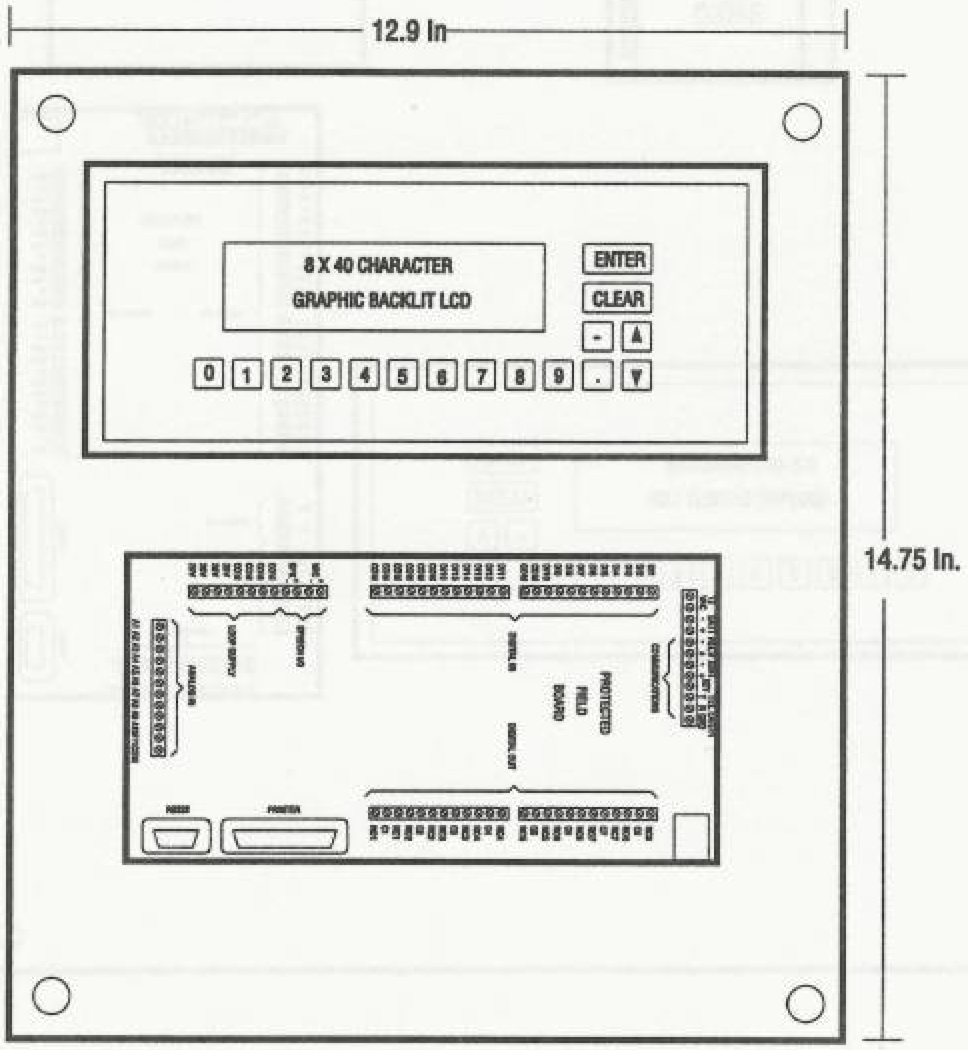


Figure A.3 RUG6 14 by 16 Backpan Mount Arrangement

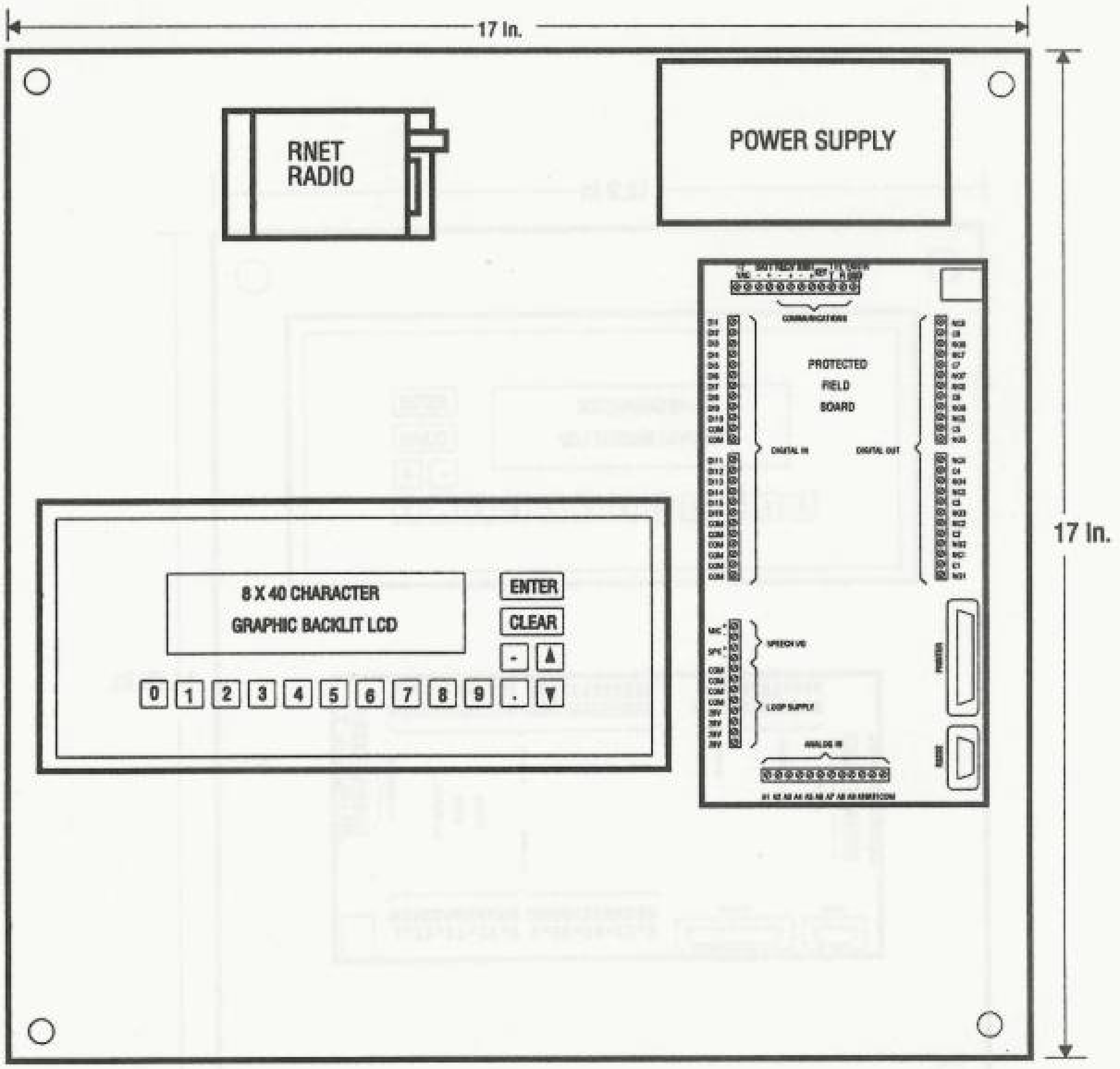


Figure A.4 RUG6 17 by 17 Backpan Mount Arrangement

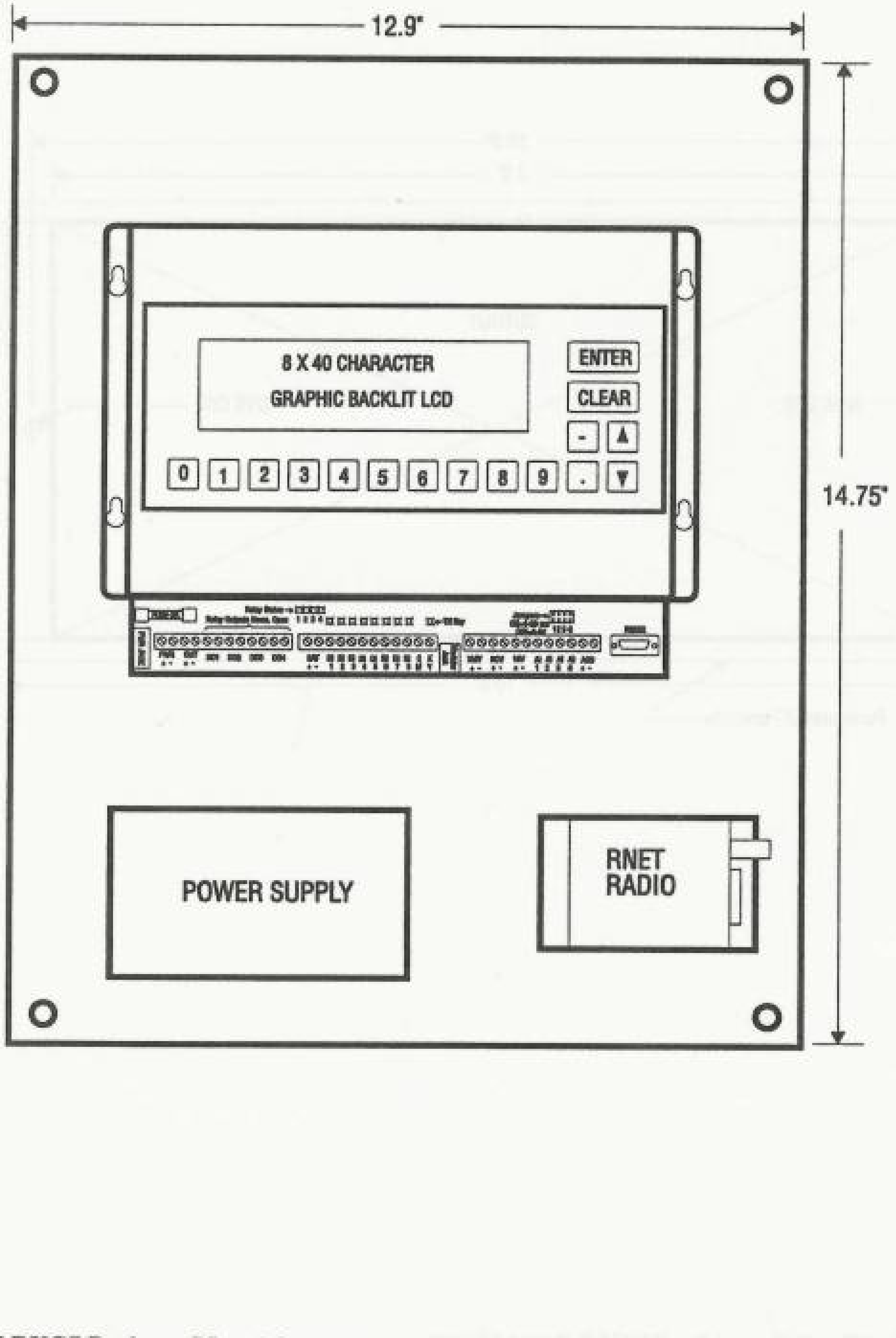


Figure A.5 RUG7 Backpan Mount Arrangement

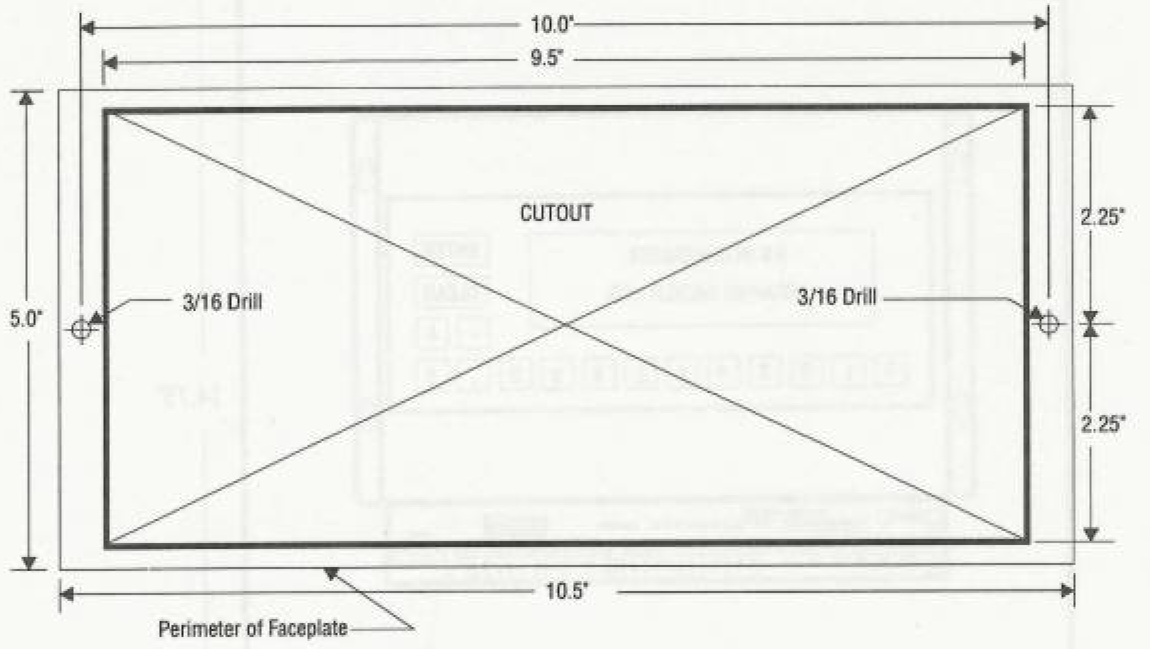


FIGURE A.6 Panel Cutout for RUG6/8 Panel Mount

APPENDIX B...EXAMPLE REMOTE RTU PROGRAM

The following program is for a RUG6D acting as an RTU in a working application. This program implements a complete RTU including analog calibration, setpoint comparison, pump controls, valve controls, display formatting, user inputs, menus, statistical data gathering, alarm detection, and more. Refer to Figure B.1 for the required I/O connections compatible with this program.

```
'SHORREM8...program for Shoreline Booster station 1
'Station address=8

'Analog inputs:
'AI%(1)=Suction pressure
'AI%(2)=Pump discharge pressure
'AI%(3)=Site discharge pressure
'AI%(4-10)=spares
'AI%(10)=unregulated supply voltage
'AI%(11)=Battery voltage

'Digital inputs:
'DI%(1)=intrusion
'DI%(2)=fire alarm
'DI%(3)=wet dry well
'DI%(4)=flow pulses
'DI%(5)=pump 1 run
'DI%(6)=pump 2 run
'DI%(7)=
'DI%(8)=
'DI%(9)=
'DI%(10)=
'DI%(11)=
'DI%(12)=
'DI%(13)=
'DI%(14)=
'DI%(15)=
'DI%(16)=Trigger tone command for test

'Digital outputs:
'DO%(1)=Open valve command
'DO%(2)=Close valve command
'DO%(3)=Transfer solenoid
'DO%(4)=Pump 1 call
'DO%(5)=Pump 2 call
'DO%(6)=
'DO%(7-8)=spares
```

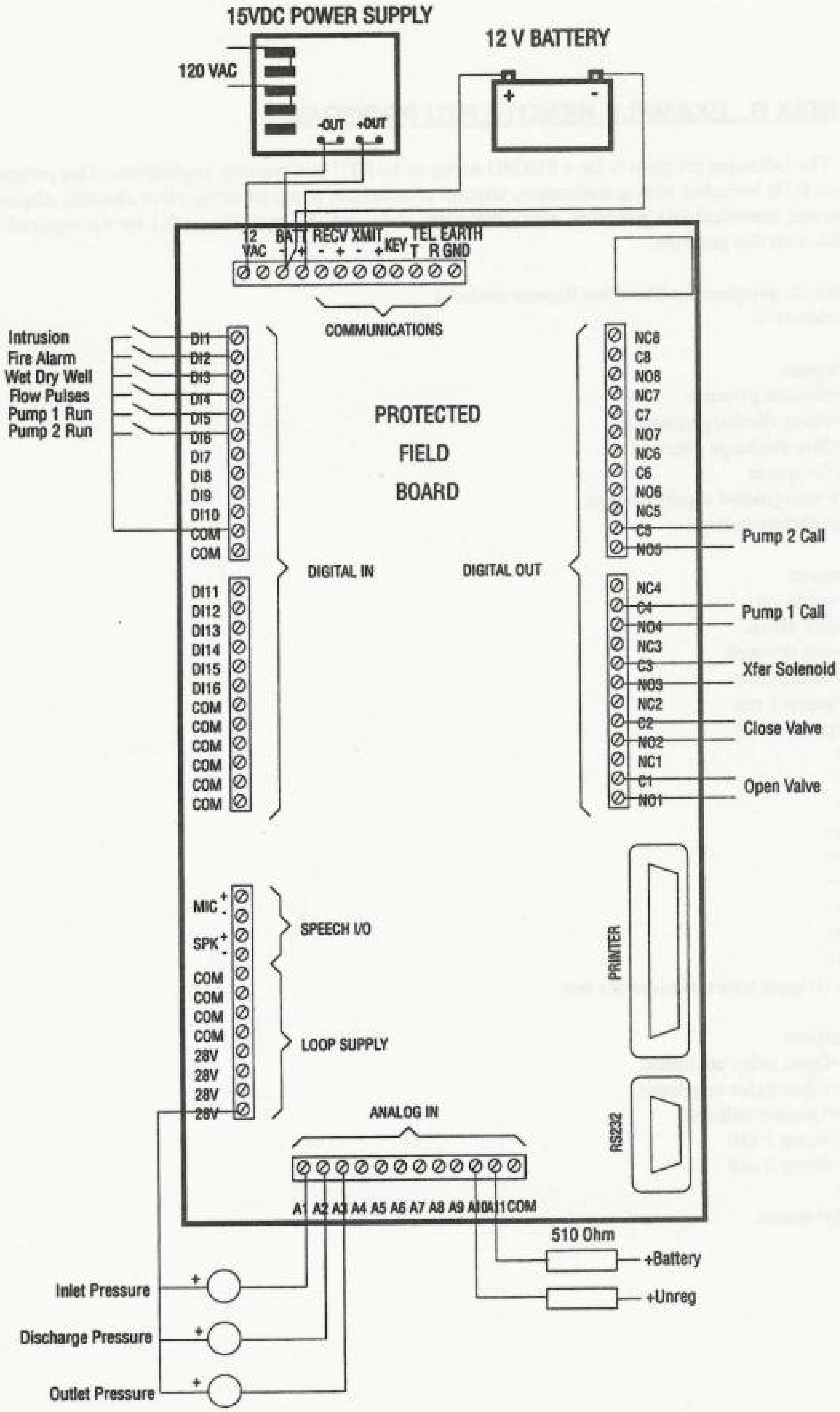


FIGURE B.1 EXAMPLE RTU PROGRAM I/O

```
'Telemetry arrays:
'AR%(SN,0)=# bytes destination to transmit
'AR%(SN,1)=bit 16= telemetry flag...always 1
'AR%(SN,2)=virtual DO's 1-16
'AR%(SN,3)=virtual DO's 17-32
'AR%(SN,4)=virtual DO's 33-48
'                         bit 14=Noon clock sync
'                         bit 15=send tone after usual msg for test
'                         bit 16=com enable
'AR%(SN,5-25)=setpoints 1-21

'AT%(SN,0)=# bytes to transmit
'AT%(SN,1)=bit 16=telemetry flag
'                         bits 1-15=spares
'AT%(SN,2)=DI%(1-16)
'AT%(SN,3)=DI%(17-32)
'AT%(SN,4)=DI%(33-48)
'AT%(SN,5)=virtual DI's to trigger alarm dialing
'                         bit 16=com fail inserted by central RUGID
'AT%(SN,6)=virtual alarms
'AT%(SN,7...)=analogs from remote:

'Setpoints:
'SP(1)=Access keyword
'SP(2)=Analog filter time constant, sec.
'SP(3)=unregulated supply threshold for power fail detect
'SP(4)=battery voltage threshold for low battery voltage alarm
'SP(5)=raw count for analog sensor fail
'SP(6)=gallons per pulse on flow input
'SP(7)=valve full traverse time, sec.
'SP(8)=valve pulse on arrow key, sec.
'SP(9)=flow control deadband, GPM
'SP(10)=close divider...shortens valve actuation time in close direction

'Timers:
'DT%(1)=general purpose timer
'DT%(2)=PID pause time, sec.
'DT%(3)=pump call delay timer, 10 sec.
'DT%(4)=Com fail counter, 60 min.
'DT%(5)=last com counter up to 4000 sec.
'DT%(6)=Flow integration timer, 10 sec.
'DT%(7-38)=virtual alarm delay timers/enables
'DT%(39)=intrusion disable timer
'DT%(40)=valve hold timer on pump transition, 60 sec.

'Misc. arrays:
'IM%(5)=image of AT%(2-5) for alarm change detection
'AY(1-11)=EU versions of analog inputs after filters
'GT()=grand totalizers
'         1=flow, 2=P1 run time, 3=P2 run time, 4=P1 starts, 5=P2 starts
'HT()=24 hr totalizers

'Misc variables:
```

```
'PA=poll address for detecting reception
'PT=poll address for transmissions
'DF=display flag...0=main, 1=master setpoints, 2=alarms
'NU=number of alarms last scan for alarm display
'NI=image of AT%(1,5) for alarm display
'UU=total flow samples accumulated
'LC=last com, 1=master, 4=3.7 MG tank
'FS=flow setpoint, from master if remote, local if local
'VC=valve control if in local...0=manual, 1=auto
'LV=3.7 MG tank level last reported
'FG=flow integration flag...0=OK, 1=kill flow rate calc til good timing

START dim AI%(11),DI%(16),DO%(8),CK%(6),SP(30)
        dim AT%(15,32),AR%(15,32),DX%(16)
        dim AM(11),AB(11),AY(11),DT%(40),G1%(16),G0%(16)
        dim IM%(6),DP%(32),PL(3),DU%(8)
        dim GT(10),HT(10)
        AI%(0)=4372                                              'Set up AI sampling
        TN=10:HU=100:TH=1000:K8=128:K3=32767:V1=32512
        DD=peek(V1+3)                                            'Get address
        K2=256*256
        gosub BINMASK                                            'Setup masks for TLM
        gosub CALINIT                                            'Init AI calibration
        DF=0:gosub CLRSCREENC:gosub DISPLAY                      'Show display
        for I=1 to 15
        AT%(I,1)=G1%(16)                                         'TLM flags
        next
        DU%(0)=0                                                         'No periodic pulse durations
        if (SP(3)+SP(2))<1 then gosub INITSP        'Init sp's?
        RA=(peek(124)*256+peek(123))-(peek(122)*256+peek(121))'RAM left
        FG=1                                                             'Kill flow rate calc til time

LOOP gosub USERIN                                                        'Watch for keystroke
        gosub TLMUPDATE                                                  'Keep outgoing data current
        gosub WATCHTIME                                                  'Watch the clock for stats
        gosub RTCAL                                                      'Handle analog inputs
        gosub ALARMS                                                     'Watch for alarms
        gosub DISPLAY                                                    'Update LCD
        gosub WATCHTEST                                                  'Watch for local tone test cmd
        gosub WATCHCOM                                                   'See if reception
        gosub CTRL                                                       'Control valve
        gosub STATS                                                      'Statistics
        goto LOOP

STATS UU=UU+1                                   'Take statistics...sample counter
        for I=1 to 2:HT(I+4)=HT(I+4)+DI%(I+4):next:return'Pump run samples

CTRL X=0:Y=AT%(1,6)                                'Y has latched alarms
        if AT%(1,5)<0 then X=1                     'Control...com fail?
        if (Y and G1%(3))<>0 then X=1              'High res.?
        if (Y and G1%(4))<>0 then X=1              'Low res?
        if (Y and G1%(8))<>0 then X=1              'Low disch pressure?
        if (Y and G1%(9))<>0 then X=1              'Low outlet pressure?
        if X=0 goto LATCHOK                        'Latched alarms OK?
        DO%(3)=0                                          'TS off
        DU%(1)=0:DU%(2)=0                                 'Kill open/close solenoids
        goto PRESSMODE
LATCHOK Y=AR%(1,2) and 4                                  'Flow or pressure mode from master?
        if (AT%(1,4) and 2)<>0 then VC=1                  'Valve auto if remote
        if Y=0 goto PRESSMODE
FLOWMODE DO%(3)=1                                         'Flow mode...Xfer solenoid on
        if FS<1 then FS=1                                 'Watch for div by zero
        AT%(1,4)=AT%(1,4) or 1                            'Show mode to master
        gosub PMPCTRL                                     'Handle pumps
        for I=1 to 2                                      'Test for pump call/run mismatch
        if DO%(I+3)<>DI%(I+4) then DT%(40)=60             'Preset valve hold timer if so
        next
        if DT%(2)<>0 or DT%(40)<>0 then return            'Pause timer still running?
        DT%(2)=AR%(1,15)                                  'Restart timer
        if VC=0 then return                               'Valve manual?
        ER=AT%(1,10)-FS                                   'Error
        if abs(ER)<SP(9) then return                      'Error > deadband?
        X=ER*(AR%(1,10)/TH)/FS*SP(7)                      'Error*PB/FS*traverse time
        if SP(10)<5 then SP(10)=5
        if X<-SP(7)/5 then X=-SP(7)/5                     'Watch range
        if X>SP(7)/SP(10) then X=SP(7)/SP(10)             'Limit pulses to traverse/5
        if X<0 goto FLOWMODE1                             'Neg. error?
        X=X*56                                            'Basic clock increments
        if X>K3 then X=K3                                 'Watch range
        DU%(2)=X:return                                   'Pulse CLOSE contact
FLOWMODE1 X=-X*56                                         'Basic clock increment
        if X>K3 then X=K3                                 'Watch range
        DU%(1)=X:return                                   'Pulse OPEN contact
PRESSMODE AT%(1,4)=AT%(1,4) and G0%(1)                    'Show mode to master
        DO%(3)=0                                          'De-energize xfer solenoid
        X=AR%(1,2)
        if DT%(3)<>0 then return                          'Pump delay timer
        if LP<1 then LP=1                                 'Lead pump range
        if LP>2 then LP=2
        XX=0:Y=AT%(1,6)                                   'Y has latched alarms
        for I=4 to 6
        if (Y and G1%(I))<>0 then XX=1                    'Latched alarm?
        next
        if XX=0 goto PRESS2                               'Skip around if OK
        for I=1 to 2:gosub PMPOFFTIME                     'Pump off
        if DT%(3)<>0 then I=3                             'Pump delay timer on?
        next:return
PRESS2 Y=DO%(4)+DO%(5)+DI%(5)+DI%(6)                      '# pumps called or running
        for I=1 to 2                                      'Test for hand off...
        if (X and G1%(I+8))<>0 goto NXPUMP                'Auto?
        if Y<>0 goto PRESS4                                  'Already one on?
        if (X and I)<>0 then gosub PMPONTIME:goto TIMTEST    'Hand?
PRESS4 if (X and I)=0 then gosub PMPOFFTIME                  'Off?
TIMTEST if DT%(3)<>0 then I=3                                'Pump delay timer on?
NXPUMP next
        if DT%(3)<>0 then return                             'Pump delay timer
        if Y<>0 goto POFFTEST                                'Have one on?
        if (X and G1%(LP+8))=0 then gosub ALTERNATE:return   'Auto?...else change
        if LV<AR%(1,5) then I=LP:gosub PMPONTIME              'Tank < SP, start pump
        return
POFFTEST if LV>AR%(1,6) then I=LP:gosub AUTOPMPOFF            'Tank > SP, stop pump
        return

AUTOPMPOFF for I=1 to 2
        if (AR%(1,2) and G1%(I+8))<>0 then gosub PMPOFFTIME   'Off if auto
        next:return

PMPONTIME if DO%(I+3)=0 then DO%(I+3)=1:DT%(3)=10:gosub ALTERNATE        'On
        return

PMPOFFTIME if DO%(I+3)<>0 then DO%(I+3)=0:DT%(3)=10                      'Pump off if on
        return

PMPCTRL if DT%(3)<>0 then return                       'Ctrl pump in flow mode...pump delay timer
        if LP<1 then LP=1                                     'Lead pump range
        if LP>2 then LP=2
        XX=0:Y=AT%(1,6)                                       'Y has latched alarms
        for I=4 to 6
        if (Y and G1%(I))<>0 then XX=1                        'Latched alarm?
        next
        if XX=0 goto PPRESS2                                  'Skip around if OK
        for I=1 to 2:gosub PMPOFFTIME                         'Pump off
        if DT%(3)<>0 then I=3                                 'Pump delay timer on?
        next:return
PPRESS2 Y=DO%(4)+DO%(5)+DI%(5)+DI%(6)                         '# pumps called or running
        X=AR%(1,2)                                            'Control bits
        for I=1 to 2                                          'Test for hand off...
        if (X and G1%(I+8))<>0 goto NNXPUMP                   'Auto?
        if Y<>0 goto PPRESS4                                  'Already one on?
        if (X and I)<>0 then gosub PMPONTIME:goto TTIMTEST    'Hand?
PPRESS4 if (X and I)=0 then gosub PMPOFFTIME                  'Off?
TTIMTEST if DT%(3)<>0 then I=3                                'Pump delay timer on?
NNXPUMP next
        if DT%(3)<>0 then return                              'Pump delay timer
        if Y<>0 then return                                   'Have one on?
        if (X and G1%(LP+8))=0 then gosub ALTERNATE:return    'Auto?...else change
        I=LP:gosub PMPONTIME                                  'Start pump if off
        return

ALTERNATE LP=LP+1                                             'Change lead
        if LP>2 then LP=1
        return

WATCHTEST if DI%(16)<1 then return                            'DI%(16) tone test trigger?
        gosub BLANK7:print "Transmitting on 4-wire port...";
        PO%=5:print "EX";                                     'Transmitter on
WATCHTEST1 if DI%(16)>0 goto WATCHTEST1                       'Loop until DI%(16) off
        gosub BLANK7:print "Test terminated..."
        PO%=5:print "Y";:return                               'Transmitter off, onhook

        GT(1)=flow, 2=P1 run time, 3=P2 run time, 4=P1 starts, 5=P2 starts
```

```
WATCHTIME if DT%(6)>2 goto WATCHT1                'Flow integration done?
WATCHLOOP if DT%(6)<>0 goto WATCHLOOP             'Wait for exact time tick
        DT%(6)=10                                 'Restart timer
        X=DX%(4):DX%(4)=0                         'Get flow pulses
        HT(1)=HT(1)+X                             'Total pulses this day
        GT(1)=GT(1)+X*SP(6)                       'Grand total gallons
        if FG<>0 then FG=0:goto WATCHT1           'Skip flow rate calc if bad timing
        X=X*SP(6)*6                               'Flow rate GPM *
        if X<0 then X=0                           'Watch range
        if X>K3 then X=K3
        AT%(1,10)=X                               'Save to TLM
WATCHT1 if MM=CK%(1) then return                  'Watch the clock
        MM=CK%(1)
        if HH=CK%(2) then return                  'New hour?
        HH=CK%(2)
        if HH<>8 then return                      '8 AM?
        X=HT(1)*SP(6)/TH                          'Total flow last 24 hrs KG *
        if X<0 then X=0                           'Watch range
        if X>K3 then X=K3
        AT%(1,12)=X:HT(1)=0                       'Save to TLM
        if UU<1 then UU=1                         'Head off divide by zero
        for I=1 to 2
        X=HT(I+4)/UU*24*HU                        'Pump run times *100
        HT(I+4)=0                                 'Clr accumulator
        if X<0 then X=0                           'Watch range
        if X>2390 then X=2400                     'Round to 24.00 hrs
        AT%(1,I+12)=X                             'Save run times to TLM
        X=DX%(4+I):DX%(4+I)=0                     'Starts
        if X<0 then X=0                           'Watch range
        if X>K3 then X=K3
        AT%(1,I+14)=X:next                        'Starts to TLM
        for I=1 to 2                                    'Pump grand totals...
        GT(1+I)=GT(1+I)+AT%(1,I+12)/HU            'Pump run times
        GT(3+I)=GT(3+I)+AT%(1,I+14)               'Pump starts
        next
        for I=1 to 5                              'Grand totals to TLM...
        if I=1 then X=GT(I)/TH                    'Total flow KG
        if I=2 or I=3 then X=GT(I)*HU             'Run times hr *100
        if I=4 or I=5 then X=GT(I)                'Starts
        Y=int(X/10000)                            'MS
        if Y<0 then Y=0                           'Watch range
        if Y>K3 then Y=K3
        AT%(1,16+I*2)=Y                           'Save MS to TLM
        Y=int(X-Y*10000)                          'Now for LS
        AT%(1,17+I*2)=Y                           'Save LS to TLM
        next
        UU=0                                      'Reset # samples
        if CK%(6)<>2 then return                  'Monday?
        for I=1 to 2:gosub PMPOFFTIME:next        'Kill pump(s)
        return

RTCAL if SP(2)<1 then SP(2)=1                     'Analog LPF & EU conversion
        Y=1/SP(2)                                 'Time constant
        for I=1 to 11
        if I=4 then I=10                                    'Skip some
        if I<9 and (AI%(I)<SP(5)) goto RTCAL2               'Sensor fail?...if so freeze
        AY(I)=(AY(I)+Y*(AI%(I)*AM(I)+AB(I)))/(1+Y)
        if AY(I)>TH then Y=TH                               'Watch for high values
        if AY(I)<0 then AY(I)=0                             'And low
RTCAL2 next
        for I=1 to 3                                        'Now for pressures
        X=AY(I)                                                  'Get pressure
        if X>K3 then X=K3                                   'Watch range
        AT%(1,I+6)=X                                        'Save to TLM
        next
        X=AY(11)*TN                                         'Battery voltage
        if X<0 then X=0
        if X>K3 then X=K3                                   'Watch range
        AT%(1,11)=X                                         'Save to TLM
        return

DISPLAY PO%=0:CC%=0:CR%=0                              'Show data on LCD
        if LC<>1 and LC<>5 then LC=1                        'Range of last com
        if DF<>0 goto DISPLAY1                              'Which display?
        print "   *** BOOSTER STATION 1, ADDR=";DD;"***"
        print
        print "3.7 MG ft=";
        if LC=4 then print AR%(LC,8)/TN;"   ";:goto DISPLX    '3.7 source?
        print AR%(LC,10)/TN;"   ";                          'Master source
DISPLX CC%=19:print "Flow GPM=";AT%(1,10);"   "
        print "Sea. PSI=";AT%(1,7);"   ";
        CC%=19:print "Flow SP GPM=";int(FS);"   "
        print "Pumps PSI=";AT%(1,8);"   ";
        CC%=19:print "Distr. PSI=";AT%(1,9);"   "
        if (AT%(1,4) and 1)=0 then print ".Pressure.";:goto DISP1
        print "...Flow...";
DISP1 if (AT%(1,4) and 2)=0 then print ".Local..";:goto DISP2
        print ".Remote.";
DISP2 if (VC)<>0 then print "..Auto..";:goto DISP3
        print ".Manual.";
DISP3 goto MENU
DISPLAY1 if DF>1 goto DISPLAY2
        print "        *** MASTER SETPOINTS ***"
        print "Bst ON ft=";AR%(1,5)/TN;"   ";
        CC%=20:print "Lo PSI in=";AR%(1,7);"   "
        print "Bst OFF ft=";AR%(1,6)/TN;"   ";
        CC%=20:print "Lo PSI dis=";AR%(1,8);"   "
        print "Flow GPM=";int(FS);"   ";
        CC%=20:print "Lo PSI out=";AR%(1,9);"   "
        print "Prop band=";AR%(1,12)/TN;"   ";
        CC%=20:print "Hi PSI dis=";AR%(1,13);"   "
        print "Pause sec=";AR%(1,15);"   ";
        CC%=20:print "Hi PSI out=";AR%(1,14);"   "
        goto MENU
DISPLAY2 gosub CLRSCREENC:print "             *** ALARMS ***"
        CR%=1:X=AT%(1,5):Y=0:YY=0
        for I=1 to 16
        if (X and G1%(I))=0 goto DSP13                              'No alarm?
        CC%=YY
        on I goto ALR1,ALR2,ALR3,ALR4,ALR5,ALR6,ALR7,ALR8
        on I-8 goto ALR9,ALR10,ALR11,ALR12,ALR13,ALR14,ALR15,ALR16
ALR1 print "Intruder ":goto DSP13X
ALR2 print "Low batt ":goto DSP13X
ALR3 print "Pwr fail ":goto DSP13X
ALR4 print "Fire     ":goto DSP13X
ALR5 print "Wet well ":goto DSP13X
ALR6 print "P 1 Fail ":goto DSP13X
ALR7 print "P 2 Fail ":goto DSP13X
ALR8 print "HiOutPrs ":goto DSP13X
ALR9 print "LoOutPrs ":goto DSP13X
ALR10 print "InXD Fail":goto DSP13X
ALR11 print "DsXD Fail":goto DSP13X
ALR12 print "OutXDFail":goto DSP13X
ALR13 print "         ":goto DSP13X
ALR14 print "         ":goto DSP13X
ALR15 print "         ":goto DSP13X
ALR16 print "Com fail ":goto DSP13X
DSP13X Y=Y+1
        if Y>5 then Y=0:YY=YY+12:CR%=1                          'Next column
DSP13 next
        X=AT%(1,6)
        for I=3 to 9
        if (X and G1%(I))=0 goto DSP13A                          'No alarm?
        CC%=YY
        on I goto ALM1,ALM2,ALM3,ALM4,ALM5,ALM6,ALM7,ALM8
        on I-8 goto ALM9,ALM10,ALM11,ALM12,ALM13,ALM14,ALM15,ALM16
ALM1 print "         ":goto DSP13Z
ALM2 print "         ":goto DSP13Z
ALM3 print "Hi Resv  ":goto DSP13Z
ALM4 print "Lo Resv  ":goto DSP13Z
ALM5 print "Hi InPres":goto DSP13Z
ALM6 print "Hi OutPrs":goto DSP13Z
ALM7 print "Lo InPres":goto DSP13Z
ALM8 print "Lo DisPrs":goto DSP13Z
ALM9 print "Lo OutPrs":goto DSP13Z
ALM10 print "         ":goto DSP13Z
ALM11 print "         ":goto DSP13Z
ALM12 print "         ":goto DSP13Z
ALM13 print "         ":goto DSP13Z
ALM14 print "         ":goto DSP13Z
ALM15 print "         ":goto DSP13Z
ALM16 print "         ":goto DSP13Z
DSP13Z Y=Y+1
        if Y>5 then Y=0:YY=YY+12:CR%=1                   'Next column
DSP13A next
MENU CC%=0:CR%=7:print "1=Reset, 2=DSP, 3=Setup, 4=Ctrl ";
        print 4000-DT%(5);"  ";
        return

WATCHCOM X=0                                          'Watch for incoming msgs
        if DT%(4)=0 then AT%(1,5)=AT%(1,5) or G1%(16) 'Com fail?
        for I=1 to 10                                 'Test for any msg
        if AR%(I,1)<0 then X=I:I=11
        next:PA=X
        if X=0 then return                                   'No message?
        LC=X                                                 'Hold last com for display & ctrls
        AR%(PA,1)=0                                          'Clr TLM flag
        if PA=1 then AT%(1,5)=AT%(1,5) and G0%(16)           'Clear com fail flag
        gosub BLANK7:print "Received from station";PA
        if PA=1 then LV=AR%(1,10)                            '3.7 MG from master
        if PA=4 then LV=AR%(4,8)                             '3.7 MG from tank sta
        if PA=1 then DT%(5)=4000:DT%(4)=3600                 'Restart ctrs
        if (AR%(1,4) and G1%(14))<>0 then CK%(1)=0:CK%(2)=12 'Noon clock sync
        AR%(1,4)=AR%(1,4) and G0%(14)                        'Clr clock sync
        if (AR%(PA,4) and G1%(15))<>0 then gosub SENDTONE    'Test tone?
        AR%(PA,4)=AR%(PA,4) and G0%(15)                      'Clr test tone
        if (AT%(1,4) and 2)=0 then return                        'Local valve mode?
        FS=AR%(1,11)                                             'Flow setpoint
        if AR%(1,2)<0 then gosub ACK:AR%(1,2)=AR%(1,2) and G0%(16)  'Master reset?
        if (AR%(1,2) and G1%(9))=0 then VC=0:return              'Auto valve?
        VC=1:return
        'Manual valve

'DT%(5)=last com counter up to 4000 sec.

SENDTONE DT%(1)=20                                               'Give 20 sec. test tone
        PO%=5:print "EX";                                        'Transmit tone on
        gosub BLANK7:print "Sending test tone to sta";PA
GOTMSLOOP if DT%(1)<>0 goto GOTMSLOOP                'Wait til done
        gosub BLANK7:print "Done with test...";
        PO%=5:print "Y";:return

BLANK3 CR%=2:goto BLNK                                           'Blank line 3
BLANK4 CR%=3:goto BLNK                                           'Blank line 4
BLANK5 CR%=4:goto BLNK                                           'Blank line 5
BLANK6 CR%=5:goto BLNK                                           'Blank line 6
BLANK7 CR%=6:goto BLNK                                           'Blank LCD line 7
BLANK8 CR%=7:goto BLNK                                           'Blank line 8
BLNK CC%=0:PO%=0:print "                    ";
        CC%=0:return

ALARMS if (AT%(1,4) and 2)<>0 goto ALARMS1           'In local control mode?
        AT%(1,5)=0:AT%(1,6)=0:return                 'Clr all alarms & exit *
ALARMS1         X=0                                  'Test alarms
        if DI%(1)<>0 and DT%(39)=0 then X=X or 1     'Intrusion?
        if AY(11)<SP(4) then X=X or 2                'Low battery?
        if AY(10)<SP(3) then X=X or 4                'Power fail?
        if DI%(2)<>0 then X=X or 8                   'Fire?
        if DI%(3)<>0 then X=X or 16                  'Wet dry well?
        if AY(3)>AR%(1,14) then X=X or G1%(8)        'Hi outlet pressure
        if AY(3)<AR%(1,9) then X=X or G1%(9)         'Lo outlet pressure
        for I=1 to 3
        if AI%(I)<SP(5) then X=X or G1%(I+9)         'Xducer fails?
        next
        for I=1 to 2                                             'Pump fails
        if DI%(I+4)<>DO%(I+3) then X=X or G1%(I+5)
        next
        X5=X:Y5=AT%(1,5)                                              'New & old alarms
        for I=1 to 16                                                 'Reset timers if alarms OK
        if DP%(I)=0 goto ALRMS55                                      'Alarm enabled?
        if (X5 and G1%(I))=0 then DT%(I+6)=DP%(I):goto ALRMS55        'Virtual alarms
        if (Y5 and G1%(I))<>0 or DT%(I+6)<>0 goto ALRMS5              'New alarm & timeout?
        Y5=Y5 or G1%(I):goto ALRMS5                                   'Set alarm
ALRMS55 Y5=Y5 and G0%(I)                                              'Clear alarm
ALRMS5 next:AT%(1,5)=Y5                                               'Save to TLM
        X=AT%(1,6):Y6=AT%(1,6)                                        'Next word of virtuals
        if LV>AR%(1,18) then X=X or G1%(3)          'Hi res.
        if LV<AR%(1,19) then X=X or G1%(4)          'Lo res.
        if AY(2)>AR%(1,13) then X=X or G1%(5)       'Hi disch pressure
        if AY(3)>AR%(1,14) then X=X or G1%(6)       'Hi outlet pressure
        if AY(1)<AR%(1,7) then X=X or G1%(7)        'Lo inlet pressure
        if AY(2)<AR%(1,8) then X=X or G1%(8)        'Lo discharge pressure
        if AY(3)<AR%(1,9) then X=X or G1%(9)        'Lo outlet pressure
        X6=X
        for I=1 to 16                                                 'Reset timers if alarms OK
        if DP%(I+16)=0 goto ALRMS66                                   'Alarm enabled?
        if (X6 and G1%(I))=0 then DT%(I+22)=DP%(I+16):goto ALRMS6     'Virtual alarms
        if (Y6 and G1%(I))<>0 or DT%(I+22)<>0 goto ALRMS6             'New alarm & timeout?
        Y6=Y6 or G1%(I):goto ALRMS6                                   'Set alarm
ALRMS66 Y6=Y6 and G0%(I)                                              'Clear alarm
ALRMS6 next:AT%(1,6)=Y6                                               'Save to TLM
        return

'SP(3)=unregulated supply threshold for power fail detect
'SP(4)=battery voltage threshold for low battery voltage alarm
'SP(5)=raw count for analog sensor fail

TLMUPDATE X=0
        for I=1 to 16
        if DI%(I)<>0 then X=X or G1%(I)                               'Setup DI's
        next
        AT%(1,2)=X                                                    'Xfer DI's to TLM
        AT%(1,1)=G1%(16)                                              'Set tlm reply flags
        return

USERIN PO%=0:get A$                                                   'Get keystroke if any
        if len(A$)=0 then return                                      'Had char?
        if A$="1" then gosub ACK:goto USEROUT                         'ACK?
        if A$="2" then gosub CHNGDSPLY:goto USEROUT                   'Change display?
        if A$="3" then gosub SETUP:goto USEROUT                       'Setup?
        if A$="4" then gosub MANCTRL:goto USEROUT                     'Local ctrl?
        if A$="D" or A$="U" then gosub VVALVCTRL:return               'Valve ctrl?
        if A$="C" then DT%(39)=3600                                   'Intrusion disable?
USEROUT gosub CLRSCREENC:gosub DISPLAY:FG=1:goto CHNG1                'Exit

VVALVCTRL if VC<>0 then return                                        'In manual?
        if (AT%(1,4) and 2)<>0 then return                            'Remote?
        X=SP(8)*56                                                    'Pulse duration
        if X<=0 then return                                           'Watch range
        if X>K3 then X=K3
        if A$="U" then DU%(1)=X:return                                   'OPEN valve
        DU%(2)=X:return                                                  'CLOSE valve

MANCTRL gosub CLRSCREENC                                                 'Enable manual control of valve
        print:print:print "Enter access keyword...":gosub KBTOX
        if X<>SP(1) then return                                          'Test keyword
MANCTRLX gosub CLRSCREENC
        print "      *** MANUAL CONTROL ***"
        print
        print "1 Flow setpoint, GPM=";FS
        print "2 Control source= ";
        if (AT%(1,4) and 2)=0 then print "Local":goto MANCTRL1
        print "Remote"
MANCTRL1 print "3 Control mode= ";
        if VC=0 then print "Manual":goto MANCTRL3
        print "Auto"
MANCTRL3 print "4 Lead pump=";LP
        print:print "Choose option to change (1-4)...";
        gosub KBTOA
        if A$="1" goto FSENTRY
        if A$="2" goto SOURCETOG
        if A$="3" goto MODETOG
        if A$="4" goto PUMPTOG
        return
FSENTRY gosub BLANK7:print "Enter new setpoint...";:gosub KBTOX
        if XX<0 or X<0 or X>K3 goto MANCTRLX
        FS=X:goto MANCTRLX                                               'Save new setpoint
SOURCETOG X=AT%(1,4) and 2                                               'Toggle source select
        if X=0 then AT%(1,4)=AT%(1,4) or 2:goto MANCTRLX                 'Turn on if off
        AT%(1,4)=AT%(1,4) and G0%(2):goto MANCTRLX                       'Turn off
MODETOG if VC=0 then VC=1:goto MANCTRLX                                  'Auto
        VC=0:goto MANCTRLX                                               'Manual
PUMPTOG DO%(4)=0:DO%(5)=0:DT%(3)=10:return                               'Pumps off...let alternate

CHNGDSPLY DF=DF+1
        if DF>2 then DF=0
CHNG1 if DF=2 then NI=-1:NU=20                                           'Force rewrite of alarms
        return

ACK if (AR%(1,2) and 4)<>0 goto ACK1    'ACK...clear lockouts
        AT%(1,4)=AT%(1,4) and G0%(1):goto ACK2                           'Pressure mode
ACK1 AT%(1,4)=AT%(1,4) or 1                                              'Flow mode
ACK2 AT%(1,5)=AT%(1,5) and G0%(16)                                       'Clr com fail latch
        DT%(4)=3600                                                      'Com fail timer
        X=AT%(1,6)                                                       'Clr level latches...
        for I=3 to 9:X=X and G0%(I):next
        AT%(1,6)=X
        DO%(1)=0:DO%(2)=0                                                'Kill open/close solenoids
        return

SETUP gosub CLRSCREENC                          'Setup setpts...
        print:print:print "Enter access keyword...":gosub KBTOX
        if X<>SP(1) then return                                          'Test keyword
        gosub CLRSCREENC
        print "        *** SETUP ***"
        print
        print "1 Local setpts   5 Alarm delay setpoints"
        print "2 Set clock      6 Initialize totalizers"
        print "3 Calibrate"
        print "4 Setup com"
        print
        print "Choose function (1-4)...";:gosub KBTOA
        if A$="1" goto LOCALSP                              'System setpoints?
        if A$="2" goto SETCLOCK                             'Set clock?
        if A$="3" goto CALIN                                'Calibrate?
        if A$="4" goto SETADDR                              'Setup com?
        if A$="5" goto ALRMDLYSP                            'Alarm delay setpoints?
        if A$="6" goto INITRT                               'Init totalizers
        return

INITRT gosub CLRSCREENC                                     'User sets run times
        print "        *** TOTALIZERS ***"
        print
        print "1 Inflow, KG=";int(GT(1)/TH*HU)/HU
        print "2 Pump 1 run time, hrs=";int(GT(2)/HU)*HU
        print "3 Pump 2 run time, hrs=";int(GT(3)/HU)*HU
        print "4 Pump 1 starts=";int(GT(4))
        print "5 Pump 2 starts=";int(GT(5))
        print "Enter total to set (1-5)...";:gosub KBTOX
        if XX<0 or X<1 or X>5 then return
        Y=X:print
        print "Enter new value...";:gosub KBTOX
        if XX<0 or X<0 then return
        GT(Y)=X:HT(Y)=0
        if Y=1 then GT(Y)=X*TH                              'Convert to gallons
        goto INITRT

ALRMDLYSP BG=1:EN=6                                         'User sets alarm delay times
ADLY1 gosub CLRSCREENC
        print "        *** ALARMS DELAYS, SEC. ***"
        for I=BG to EN:J=I:gosub ALRMNAMEJ                  'Get alarm name
        print I;A$;DP%(I):next
        print "Choose setpoint to change...";:gosub KBTOX
        if (XX<0) or (X<BG) or (X>EN) goto ADLY2            'In range?
        print:Y=X
        print "Enter new value for SP";X;"...";:gosub KBTOX
        if XX<0 or X<0 or X>32767 goto ADLY2                'In range?
        DP%(Y)=X:goto ADLY1                                 'Save & go for more
ADLY2 BG=BG+6:EN=BG+5                                       'Point to next group
        if BG>32 then return
        if EN>32 then EN=32
        goto ADLY1

ALRMNAMEJ if J<1 or J>32 then A$="":return      'Get alarm name I to A$
        on J goto AL1,AL2,AL3,AL4,AL5,AL6,AL7,AL8
        on J-8 goto AL9,AL10,AL11,AL12,AL13,AL14,AL15,AL16
        on J-16 goto AL17,AL18,AL19,AL20,AL21,AL22,AL23,AL24
        on J-24 goto AL25,AL26,AL27,AL28,AL29,AL30,AL31,AL32
        A$="":return
AL1 A$="Intrusion      ":return
AL2 A$="Low battery    ":return
AL3 A$="Power fail     ":return
AL4 A$="Fire alarm     ":return
AL5 A$="Wet dry well   ":return
AL6 A$="Pump 1 fail    ":return
AL7 A$="Pump 2 fail    ":return
AL8 A$="High outlet pres":return
AL9 A$="Low outlet press":return
AL10 A$="Inlet XD fail  ":return
AL11 A$="Disch XD fail  ":return
AL12 A$="Outflow XD fail ":return
AL13 A$="":return
AL14 A$="":return
AL15 A$="":return
AL16 A$="":return
AL17 A$="":return
AL18 A$="":return
AL19 A$="High res. alarm ":return
AL20 A$="Low res. alarm  ":return
AL21 A$="Hi disch press  ":return
AL22 A$="Hi outlet press ":return
AL23 A$="Low inlet press ":return
AL24 A$="Low disch press ":return
AL25 A$="Low outlet press":return
AL26 A$="":return
AL27 A$="":return
AL28 A$="":return
AL29 A$="":return
AL30 A$="":return
AL31 A$="":return
AL32 A$="":return
LOCALSP gosub CLRSCREENC                                   'User sets system setpoints
        print "    *** LOCAL SETPOINTS pg 1 ***"
        print
        print "1 Access keyword=";SP(1)
        print "2 Analog filter sec.=";SP(2)
        print "3 Power fail V=";SP(3)
        print "4 Low battery V=";SP(4)
        print "5 Analog XD fail raw count=";SP(5)
        print "Choose setpoint to change (1-5)...";:gosub KBTOX
        if XX<0 or X<1 or X>5 goto LOCALSP1                'In range?
        Y=X:gosub BLANK8
        print "Enter new value for SP";Y;"...";:gosub KBTOX    'Get new value
        if XX<0 or X<0 goto LOCALSP                        'In range?
        SP(Y)=X:goto LOCALSP                               'Save it
LOCALSP1 gosub CLRSCREENC                                  'User sets system setpoints
        print "    *** LOCAL SETPOINTS pg 2 ***"
        print
        print "6 Flow gallons/pulse=";SP(6)
        print "7 Valve full traverse sec.=";SP(7)
        print "8 Valve pulse on key, sec.=";SP(8)
        print "9 Flow deadband, GPM=";SP(9)
        print "10 Close divider (10-50)=";SP(10)
        print "Choose setpoint to change (6-10)...";:gosub KBTOX
        if XX<0 or X<6 or X>10 then return                          'In range?
        Y=X:gosub BLANK8
        print "Enter new value for SP";Y;"...";:gosub KBTOX          'Get new value
        if XX<0 or X<0 goto LOCALSP1                                'In range?
        SP(Y)=X:goto LOCALSP1                                       'Save it

SETADDR V1=32512:V2=V1+16:V3=V2+16                                  'Start of voted table
        gosub CLRSCREENC
        gosub OPENRAM:X=3:gosub DELAYX                              'Open RAM & wait for write protect
        gosub ADDRESS                                               'Set address
        gosub SETCRCOPS                                             'Setup for CRC radio TLM
REBOOT print "REBOOTING...PLEASE WAIT":X=2:gosub DELAYX
        poke 4,3:poke 5,136:X=USR(0)                                'Jmp to $8803 (MAIN)

DELAYX DT%(1)=X                                                     'Delay X seconds
DELAY1 if DT%(1)<>0 goto DELAY1
        return

ADDRESS X=8:print:print "Setting address to";X
        poke V1+3,X:poke V2+3,X:poke V3+3,X:return                  'Store new unit address

VOTEXTOY poke V1+Y,X:poke V2+Y,X:poke V3+Y,X:return                 'Poke X to VOTDAT,Y...

OPENRAM print "Opening RAM..."
        X=peek(969) and 191:poke 969,X:poke 34800,X:return          'Open RAM

SETCRCOPS print "Setting up CRC flags & com delays..."
        X=1:Y=1:gosub VOTEXTOY                                      'Setup RAM for CRC, radio..parity
        X=22:Y=2:gosub VOTEXTOY                                     'Baud, stop bits, word length
        X=57:Y=10:gosub VOTEXTOY                                    'Tone use...low tones
        X=9:Y=11:gosub VOTEXTOY                                     'CRC communications
        X=7:Y=12:gosub VOTEXTOY                                     'Delay=.1 sec before XMIT
        X=64:Y=15:gosub VOTEXTOY                                    'Setup modem connection
        return

BINMASK X=1  'Setup mask fields
        for I=1 to 15:G1%(I)=X:G0%(I)=-X-1:X=X*2:next
        G1%(16)=-32768:G0%(16)=32767:return

SETCLOCK gosub TIMENOW:print "Enter new day of week:"
        print "1=Sun, 2=Mon, 3=Tue, 4=Wed"
        print "5=Thu, 6=Fri, 7=Sat";:gosub KBTOA
        if A$<"1" or A$>"7" goto MONTHIN
        CK%(6)=val(A$)
MONTHIN gosub TIMENOW:print "Enter new month (1-12):":input I
        if I<1 or I>12 goto DAYOFMONTH
        CK%(4)=I
DAYOFMONTH gosub TIMENOW:print "Enter day of month (1-31):":input I
        if I<1 or I>31 goto YEARIN
        CK%(3)=I
YEARIN gosub TIMENOW:print "Enter new year (0-99):":input I
```

```
        if I<0 or I>99 goto HOURIN
        CK%(5)=I
HOURIN gosub TIMENOW:print "Enter new hour (0-23):":input I
        if I<0 or I>23 goto MINUTEIN
        CK%(2)=I
MINUTEIN gosub TIMENOW:print "Enter new min. (0-59):":input I
        if I<0 or I>59 then return
        CK%(1)=I:CK%(0)=0:return

TIMENOW gosub RTC
        print:print "Present date/time:";JJ$:return

'**********Setup RTC string in JJ$:**********

RTC I=CK%(6)
  if I<1 or I>7 then JJ$="   ":goto SKIPDAY
  I=3*(CK%(6)-1)+1:C0=CK%(1)
  JJ$=mid$("SUNMONTUEWEDTHUFRISAT",I,3)+" "         'Day of week
SKIPDAY I=4:gosub TIMEX :JJ$=JJ$+"/"                'Month
  I=3:gosub TIMEX :JJ$=JJ$+"/"                      'Day of month
  I=5:gosub TIMEX :JJ$=JJ$+" "                      'Year
  I=2:gosub TIMEX :JJ$=JJ$+chr$(58)                 'Hours:
  I=1:gosub TIMEX :JJ$=JJ$+" ":return               'Minutes
TIMEX A$=str$(CK%(I))                               'read clock & format to 2 chars.
  IF len(A$)<3 then JJ$=JJ$+"0"+right$(A$,1):return
  JJ$=JJ$+right$(A$,2):return

'KBTOX reads keyboard and puts value in X
'Allows 120 sec for keystroke
'At end, if error or CR only, XX=-1, otherwise X has value entered

KBTOX J=CC%:DT%(1)=120:X=0:XX=0:I=0:L=0:B$=""       'Get KB value to X
KBTOX1 if DT%(1)=0 then XX=-1:return
        get A$                                      'Get character
        if len(A$)=0 goto KBTOX1
        DT%(1)=120                                  'Restart timer on keystroke
        if A$="C" or A$="D" or A$="U" goto CLRIT
        if X<0 and A$="-" goto CLRIT                '2 "-" signs?
        if L<>0 and A$="." goto CLRIT               '2 decimals?
        I=I+1:print A$;                             'Count chars received & echo
        if A$="-" then X=-1                         'Flag "-"
        if I=1 and A$=chr$(13) then XX=-1:return   'CR only?
        if A$=chr$(13) then X=val(B$):return
        B$=B$+A$:goto KBTOX1                        'Save char & go for more
CLRIT CC%=J:for K=1 to I:print " ";:next:CC%=J:goto KBTOX   'Clr entry

KBTOA DT%(1)=120                                    'Wait up to 120 sec. for keystroke
KB1 if DT%(1)=0 then A$="C":return                  'Timeout?
        get A$
        if len(A$)=0 goto KB1
        return                                      'Return with keystroke in A$

'CALMAN calibration routines
'Interacts with user to gather coefficients for 2 point calibration
```

```
'For analog input calibration:
'       M and B are stored in AM() and AB() respectively
'       To avoid losing cal when editing program, the calibration
'       constants are stored in the reserved area from
'       $1F00 through $1FFF.  These are loaded by CALINIT upon
'       boot up.  A total of 32 analog channels can be stored in this
'       area.  This routine allocates space for 24 analog inputs, and
'       8 analog outputs.  They are arranged thus:
'               $1F00-$1F03=AM(1)*65536
'               $1F04-$1F07=AB(1)*65536
'               $1F08-$1F0B=AM(2)*65536
'                       .
'                       .
'               $1FC0-$1FC3=OM(1)*65536
'               $1FC4-$1FC7=OB(1)*65536

'In realtime operation, result would be:
'       result(I)=AM(I)*AI%(I)+AB(I) for inputs 1 to 11
'       result(I)=AM(I)/AI%(I)+AB(I) for inputs 12 and up.

'For analog output calibration:
'       M and B are stored in OM() and OB() respectively

'In realtime operation, result would be:
'       AO%(I)=OM(I)*value(I)+OB(I)

'CALINIT reads reserved user area and gathers saved cal coefficients.
'       Should be executed at beginning of program.

CALINIT for I=1 to 11
        II=(I-1)*8:gosub CALRAMINX                      'Get AM() cal value to X
        AM(I)=X/65536                                   'Scale down and save
        II=(I-1)*8+4:gosub CALRAMINX                    'Get AB() cal value to X
        AB(I)=X/65536                                   'Scale down & save
        next
        for I=1 to 8
        II=(I+23)*8:gosub CALRAMINX                     'Get OM() cal value to X
        OM(I)=X/65536                                      'Scale down & save
        II=(I+23)*8+4:gosub CALRAMINX                   'Get OB() cal value to X
        OB(I)=X/65536
        next
        return

CALRAMINX J=7936+II                             'Get X from reserved area, byte J
        X=peek(J)+256*peek(J+1)+65536*peek(J+2)
        XX=peek(J+3) and 128                    'Get sign
        II=peek(J+3) and 127                    'Get MS byte, no sign
        X=X+16777216*(II)
        if XX=0 then return
        X=-X:return

CALRAMOTX J=7936+II:I1%=0                       'Save X to location in reserved area, byte J
        if X<0 then I1%=128:X=-X                'Get sign of X
        X=X*65536:XX=int(X/16777216)
        X1%=XX and 127 or I1%:poke J+3,X1%               'MS byte (1)
        X=X-XX*16777216:X1%=int(X/65536):poke J+2,X1%    'Byte (2)*****
        X=X-X1%*65536:X1%=int(X/256):poke J+1,X1%        'Byte (3)*****
        X=X-X1%*256:poke J,int(X):return                 'LS byte (4)

CALIN gosub CLRSCREENC
        print "Enter input channel to calibrate."
        print "Allowed range is 1 through 11, 0 exits":input X
        if X<1 or X>11 goto CALDONE
        CH=int(X)                                        'Save channel we're calibrating
        print "Apply known low value to analog input";CH
        print "Key in corresponding engrg. units value.":input X
        A1=X:A2=AI%(CH)                                  'Save it and A/D output
        print "Apply known high value to analog input";CH
        print "Key in corresponding engrg. units value.":input X
        A3=X:A4=AI%(CH)                                  'Save it and A/D output
        if CH>11 then A2=1/A2:A4=1/A4                    '1/AI%(I) if expansion A/D
        if abs(A2-A4)>.000001 goto COMPUT
        print "Input value error...restarting cal."
        for I=1 to 1000:next                             'Delay
        goto CALIN
COMPUT M=(A1-A3)/(A2-A4)                                 'Compute M
        B=A1-M*A2                                        'and B

'Now show result of this calibration
        gosub CLRSCREENC                                 'Clear the LCD
        A2=100:A3=0.5                                    'Use for significance control
CALOOP CR%=0:CC%=0                                       'Set display cursor location
        print "Present AI";CH;"=";AI%(CH);"  "           'Show user A/D input
        if CH<12 then A1=int((M*AI%(CH)+B)*A2+A3)/A2     'Control significance
        if CH>11 then A1=int((M/AI%(CH)+B)*A2+A3)/A2     'Expansion AI's
        print "Engrg. value=";A1;"  "                    'Show resulting value
        print "Key in 1 to save"
        print "Key in 2 to discard"
CAL1 get A$
        if len(A$)=0 goto CALOOP                         'Watch for keystroke
        if A$<>"1" goto CALIN
        AM(CH)=M:AB(CH)=B                                'Save coefficients
        X=AM(CH):II=(CH-1)*8:gosub CALRAMOTX             'Put in reserved area
        X=AB(CH):II=(CH-1)*8+4:gosub CALRAMOTX           'Ditto
        goto CALIN                                       'Return for more

CALDONE print "Exiting calibration routines."
        for I=1 to 1000:next:return                      'Delay & exit

CLRSCREENC poke 549,128:PO%=0:print chr$(12):CC%=0:CR%=0:return

CLRSCREENG poke 549,0:PO%=0:print chr$(12):CC%=0:CR%=8:return

INITSP for I=1 to 32:DP%(I)=1:next                       'Alarm enables & delays
        DP%(17)=0:DP%(18)=0                              'Kill some statuses
        SP(2)=1:SP(3)=13:SP(4)=12:SP(5)=100
        SP(6)=10:SP(7)=7:SP(8)=.2:SP(9)=100:SP(10)=10:return
```

```
'SP(1)=Access keyword
'SP(2)=Analog filter time constant, sec.
'SP(3)=unregulated supply threshold for power fail detect
'SP(4)=battery voltage threshold for low battery voltage alarm
'SP(5)=raw count for analog sensor fail
'SP(6)=gallons per pulse on flow input
'SP(7)=valve full traverse time, sec.
'SP(8)=valve pulse on arrow key, sec.
'SP(9)=flow control deadband, GPM
'SP(10)=close divider
```

APPENDIX C...EXAMPLE AUTODIALING CENTRAL PROGRAM

This program is the central program from a working polled system having autodialing, MODBUS interface, dialin reporting, local alarms, remote alarms, and more. Refer to figure C.1 for required I/O.

```
'SHORCEN...Central program for Shoreline system, address=1
'Master station address=1
'Remote Station address=3-20

'Analog inputs:
'AI%(1-11)=spares

'Digital inputs:
'DI%(1-14)=Spares
'DI%(15)=simulate ringing for test
'DI%(16)=tone on command for test

'Digital outputs:
'DO%(1)=Common alarm (remotes and master)
'DO%(2)=
'DO%(3)=
'DO%(4-8)=spares

'Linkage from master RUGID to SCADAVU:
'100 words per station:
'        50 receive words...X0 to X49
'        50 transmit words..X50 to X99

'Telemetry arrays, sent by master, received by remote:

'Master commands:
'AT%(1,4)
'        bit 1=dialer ACK
'        bit 2=dial all remotes (full poll cycle.)
'AT%(1,6)=
'AT%(1,7)=Poll interval, sec.
'AT%(1,8)=

'AT%(SN,0)=# bytes destination to transmit
'AT%(SN,1)=bit 16= telemetry flag...always 1
'AT%(SN,2)=virtual DO's 1-16
'AT%(SN,3)=virtual DO's 17-32
'AT%(SN,4)=virtual DO's 33-48
'                          bit 14=Sync clocks (remotes set clocks to 12:00)
'                          bit 15=Command remote tone after usual message
'                          bit 16=com enable
'AT%(SN,5-25)=setpoints 1-21
```

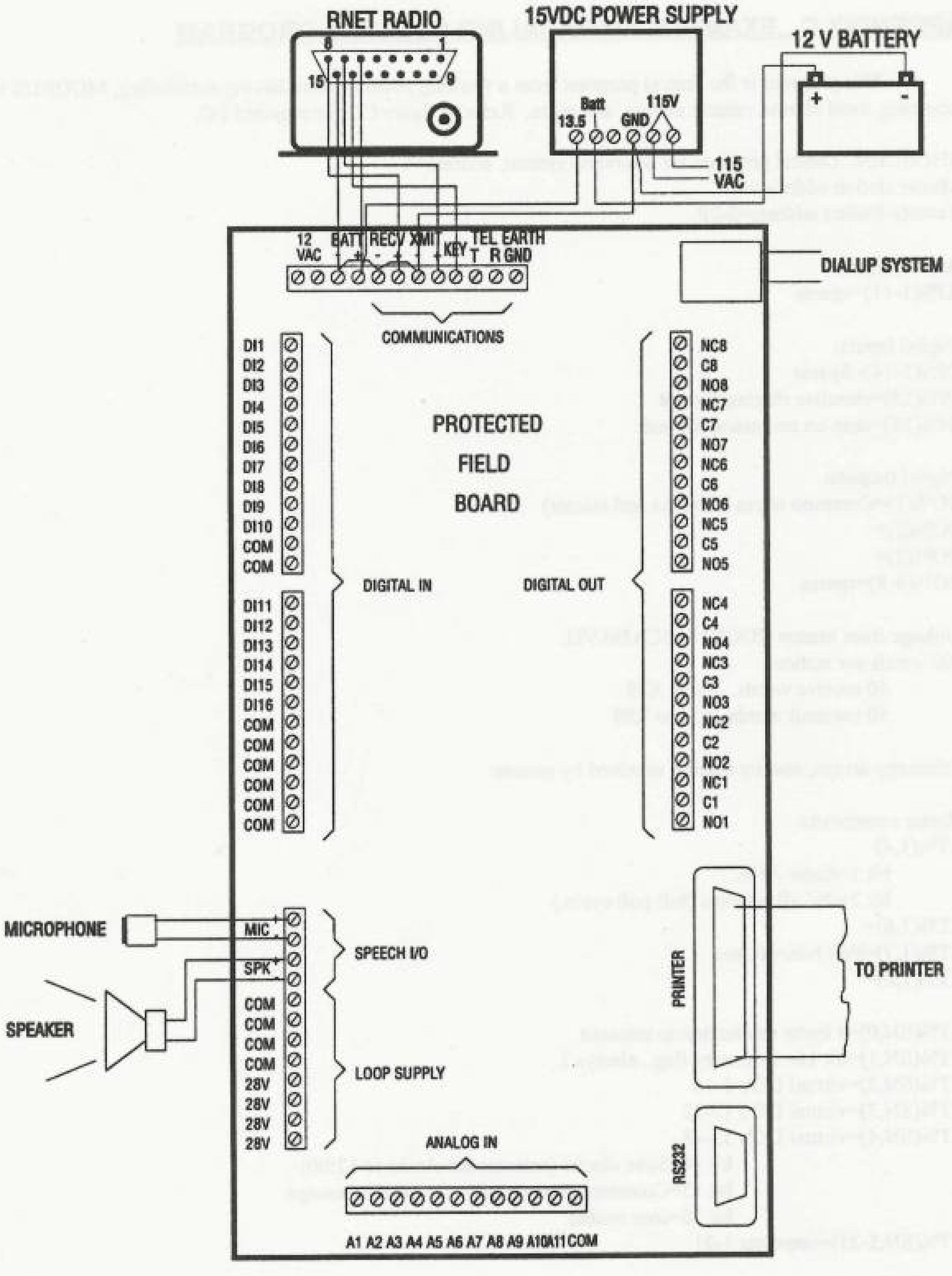


FIGURE C.1 EXAMPLE MASTER I/O

```
'AR%(SN,0)=# bytes to transmit
'AR%(SN,1)=bit 16=telemetry flag
'                         bits 1-15=spares
'AR%(SN,2)=DI%(1-16)
'AR%(SN,3)=DI%(17-32)
'AR%(SN,4)=DI%(33-48)
'AR%(SN,5)=virtual DI's to trigger alarm dialing
'                         bit 16=com fail inserted by central RUGID
'AR%(SN,6)=virtual DI's that don't trigger dialing
'AR%(SN,7-25)=analogs from remote, 1-20

'Timers:
'DT%(1)=general purpose timer
'DT%(2)=
'DT%(3)=Com reply timer
'DT%(4)=Speech redial delay timer, 120 min.
'DT%(5)=poll cycle timer
'DT%(6)=master fail timer...5 min.
'DT%(7)=polling delay timer, random
'DT%(8)=delay timer for dialing next operator, 1 min.
'DT%(9)=delay for local power fail alarm

'Misc. arrays:
'IM%(30,2)=alarm image to watch for changes
'OL%(30)=station on line flags...1= on line; 2= need to poll
'CF%(30)=station com fail counters
'AS$(30,15)=Alarm annunciation strings
'PH$(8)=dialer phone numbers
'CO$(8)=dialer security codes
'GT(10)=grand totals calculated by master

'Misc variables:
'AK=alarm ACK flag...0=OK, 1=local annun., 2=already dialed this cycle
'LA=last address sampled by master (for master com fail)
'MF=master fail flag...0=ok
'DE=dialer enable...0=OFF, 1=local, 2=dial
'DP=display we're on...0=main, 1=AT%(), 2=AR%()
'RG=# rings to answer
'SC=security code used on callin
'AF=alarm flag, 0=no alarms to report
'PA=poll address for rcv watcher
'PT=poll address for transmit
'CY=flag for poll cycle timing: 0=random 20-80 sec, 1=short cycles

'Speech rqd:
'          1-10=numerics "1, 2, 3...0"
'          11="point"
'          12="minus"
'          13="This is the Shoreline autodialer."
'          14="Please enter your security code followed by star key."
'          15="Your security code..."
'          16="...is accepted."
'          17="...is rejected."
'          18="Thank you...goodbye."
'          19="Hit any key to stop report."
'          20="repeating..."
'          21="There are presently no alarms."
'                   22="Select station from following list..."
'                   23="...is on"
'                   24="...is off"
'                   25="...is auto"
'          26="...has failed"
'                   27="The level is..."
'                   28="...feet"
'                   29="Enter selection followed by the star key"
'                   30="...on..."
'                   31="...off..."
'          32="...high alarm..."
'          33="...low alarm..."
'          34="discharge pressure..."
'          35="Suction pressure..."
'          36="Press [1] for alarm report, [2] to acknowledge alarms"
'          37="outlet pressure..."
'                   38="Select pump, booster or well number..."
'          39="Fire alarm is on."
'          40="Dry well is wet"
'          41="Valve has failed"
'                   42="...is running."
'                   43="...is not running."
'          44="Transducer has failed..."
'                   45="...is called."
'          46="The computer has failed."
'          47="0.4 MG tank..."
'          48="3.7 MG tank..."
'          49="2.0 MG tank..."
'          50="Excessive pump calls have occured."

'          51="Power has failed"
'          52="Battery voltage is low"
'          53="Pressure..."
'          54="...is low"
'          55="...is high"
'          56="Communications..."
'          57="Flow..."
'          58="...sensor...
'          59="...level..."
'          60="Intruder is present."
'          61="pump..."
'          62="booster..."
'          63="Pump 1..."
'          64="Pump 2..."
'          65="sump level..."
'          66=""
'          67="reservoir..."
'          68="Inflow..."
'          69="Outflow..."
'          70=""
'           71="tank..."
'Station 3 pump names:
'Station 4 pump names:
'           80="Reservoir booster 1"
'           81="Reservoir booster 2"
'           82=""
'           83=""
'           84=""
'           85=""
'           86=""
'           87=""
'Station 5 pump names:
'           88=""
'           89=""
'           90=""
'           91=""
'           92=""
'           93=""
'           94=""
'           95=""
'Station 6 pump names:
'           96=""
'           97=""
'           98=""
'           99=""
'           100=""
'           101=""
'           102=""
'           103=""
'Station 7 pump names:
'           104=""
'           105=""
'           106=""
'           107=""
'           108=""
'           109=""
'           110=""
'           111=""
'Station 8 Pump names:
'           112="Booster station 1, pump 1"
'           113="Bosoter station 1, pump 2"
'           114=""
'           115=""
'           116=""
'           117=""
'           118=""
'           119=""
'Station 9 pump names:
'           120="Booster station 2, pump 1"
'           121="booster station 2, pump 2"
'           122=""
'           123=""
'           124=""
'           125=""
'       126=""
'       127=""
'Station 10-15 pump names=strings 128-174
'       201="At central site..."
'       202=""
'       203=""
'       204="At the reservoir booster station..."
'       205="At the 2 MG reservoir..."
'       206="At supply station 3..."
'       207="At supply station 1..."
'       208="At booster station 1..."
'       209="At booster station 2..."
'       210=""

START dim AI%(11),DI%(16),DO%(8),CK%(6)
        dim AT%(15,32),AR%(15,32),IM%(15,2)
        dim DT%(10),G1%(16),G0%(16),CN%(15),GT(10)
        dim OL%(15),CF%(15),AS$(15,32),PH$(8),CO$(8)
        dim AY(11),AM(11),AB(11)
        AI%(0)=4372                                         'Poll rate
        gosub CALINIT                                       'Init calibration
        gosub BINMASK                                       'Setup masks for TLM
        for I=1 to 15:CF%(I)=10                             'Init com fail, clr need flags
        if (AT%(1,4)<0) then OL%(I)=1                       'On line?
        AT%(I,1)=G1%(16)                                    'Set TLM flags
        IM%(I,0)=0:IM%(I,1)=0:IM%(I,2)=0                    'Clear alarm images
        next                                                'Init TLM flags
        TN=10:HU=100:TH=1000:K8=128:K3=32767
        V1=32512:V2=V1+16:V3=V2+16                          'Start of voted table
        K1=1062:K2=256*256                                  'Rcv port address
        DD=peek(V1+3)                                       'Get address
        PO%=5:print "E";                                    'Modem to 4-wire
        gosub CLRSCREEN:gosub DISPLAY                       'Show display
        AK=0                                                'Kill dialing *
        DT%(6)=300:MF=0                                     'Master fail flag
        RA=(peek(124)*256+peek(123))-(peek(122)*256+peek(121))  'RAM left
        CY=0                                                'Random poll cycling
        for I=4 to 9:AT%(I,4)=AT%(I,4) or G1%(16):next      'Poll all

LOOP gosub POLL                                             'Poll remotes
        gosub USERIN                                        'Watch for user inputs
        gosub WATCHCOM                                      'Watch for incoming com
        gosub WATCHCPU                                      'Watch for CPU fail & command
        gosub WATCHDIAL                                     'See if need to dial
        gosub WATCHRING                                     'Watch for incoming ringing
        gosub WATCHTIME                                     'Watch the clock
        gosub WATCHTEST                                     'Watch for tone test cmd
        gosub ALARMS                                        'Watch for local alarms
        gosub ANYALARMS                                     'See if any alarms
        gosub RTCAL
        goto LOOP

RTCAL AY(10)=AI%(10)*AM(10)+AB(10)                          'Battery voltage
        return
```

```
ALARMS X=AY(10)*TN                                                'Battery voltage
        Y=AR%(2,5)                                                'Alarm word
        if X>AT%(2,6) then DT%(9)=60:goto ALRMOFF                 'Alarm OK?
        if DT%(9)=0 then AR%(2,5)=Y or 1:goto ALARMS1             'Alarm on
ALRMOFF AR%(2,5)=Y and G0%(1)                                     'Alarm off
ALARMS1 if Y=AR%(2,5) then return                                 'Change?
        SN=2:gosub STATTEST:return                                'Watch for change

WATCHRING if RG=0 then poke 974,0                                 'Disable autoanswer if RG=0
        if DI%(15)<>0 goto WATCHRRG                               'Watch for simulated ring
        if (RG=0 or peek(974)<RG) then return                     'Watch for ringing
WATCHRRG PO%=5:print "EL"                                         'Go offhook, spch to 2-wire
        gosub BLANK7:print "Incoming call...";
WATCHRR6 gosub CLRTTONES                                          'Clear tone input buffer
WATCHRR3 PO%=4:print "P13";:gosub SPKWAIT                         '"This is ...autod
WATCHRR7 PO%=4:print "P36";:gosub SPKWAIT                         '"Select 1 for alrms, 2 ACK
        poke 1146,255                                             'Retriggr tlm watchdog
        gosub CLRTTONES:gosub KBTOATT
        if A$="1" then gosub ALRMRPT:goto WATCHRR7                'Alarm report?
        if A$="2" then gosub SECURANY:goto WATCHRR7               'Security for ACK?
WATCHROUT gosub GOODBYE:gosub HANGUP:gosub BLANK7:return

SECURANY if SC=0 goto SECURACK                                    'Need security code?
        PO%=4:print "P14";:gosub SPKWAIT                          '"Enter security code..."
        gosub KBTOXTT                                             'Get code
        if X=714 goto SECURACK                                    'Our code
        if X<>SC then PO%=4:print "P15P17";:gosub SPKWAIT:return  '"Rejected"
SECURACK PO%=4:print "P15P16";:gosub SPKWAIT                      '"Accepted"
        gosub ACK:return                                          'ACK alarms

ALRMRPT if AF=0 then PO%=4:print "P21";:gosub SPKWAIT:return      'No alarms?
        RR=1:gosub VERBOUT:return                                 'Report them

WATCHTEST if DI%(16)<1 then return                                'DI%(16) tone test trigger?
        gosub BLANK7:print "Transmitting on 4-wire port...";
        PO%=5:print "EX";                                         'Transmitter on
WATCHTEST1 if DI%(16)>0 goto WATCHTEST1                           'Loop until DI%(16) off
        gosub BLANK7:print "Test terminated..."
        PO%=5:print "Y";:return                                   'Transmitter off

WATCHTIME if MM=CK%(1) then return                                'Watch the RTC
        MM=CK%(1)
        X=MM+60*CK%(2)                                            'Minutes since midnight
        if X>(10+60*8) and RF=0 then RF=1:gosub CALCTOTAL         'Totals at 8:10
        if HH=CK%(2) then return                                  'New hour?
        HH=CK%(2)
        if HH=12 then gosub TIMESYNC                              'Sync all remote clocks
        if HH=15 then RF=0                                        'Reset totalization flag
        return

CALCTOTAL for I=1 to 8                                            'Calc totals
        if I<5 then GT(I)=GT(I)+AR%(4,14+I)                       'Flows Kgal
        if I=5 or I=6 then GT(I)=GT(I)+AR%(4,14+I)/HU             'Run times
        if I>6 then GT(I)=GT(I)+AR%(4,14+I)                                'Starts
        X=GT(I)                                                            'Total flow MG
        Y=int(X/10000)                                                     'MS
        if Y<0 then Y=0                                                    'Watch range
        if Y>K3 then Y=K3
        AR%(1,I*2)=Y                                                       'Save MS to TLM
        Y=int(X-Y*10000)                                                   'Now for LS
        AR%(1,1+I*2)=Y                                                     'Save LS to TLM
        next:return

TIMESYNC for I=3 to 10                                                     'Sync remote clocks at noon
        AT%(I,4)=AT%(I,4) or G1%(14)                                       'Set sync bits for remotes
        next
        return                                                             'Init polling

WATCHCPU if DT%(6)<>0 goto WATCHCPU1                                       'Watch for CPU fail
        if MF<>0 goto WATCHCPU1                                            'Already have fail?
        MF=1:AR%(2,5)=AR%(2,5) or G1%(16):SN=2:gosub STATTEST               'Master fail?
WATCHCPU1 if LA=peek(K1) goto WATCHCPU2                                    'New master address
        LA=peek(K1):PO%=0:CC%=220:CR%=64:print LA;                         'Show master poll
        DT%(6)=300:MF=0
        AR%(2,5)=AR%(2,5) and G0%(16):SN=1:gosub STATTEST                  'Clr master fail flag
WATCHCPU2 if (AT%(1,4) and G1%(2))=0 goto WATCHCPU3                        'Master poll flag?
        AT%(1,4)=AT%(1,4) and G0%(2)                                       'Clr the flag
WATCHCPU3 X=AT%(1,4)                                                       'Master flags
        if (X and 1)<>0 then gosub ACK:AT%(1,4)=X and G0%(1)               'Silence?
        X=peek(544)                                                        'Address written by MODBUS
        if X=0 then return                                                 'Nothing new
        gosub MASTXFER4                                                    'Distrib sta 4 setpoints
        gosub MASTXFER5                                                    'Distrib sta 5 setpoints
        poke 544,0:return                                                  'Ready for next

MASTXFER4 X=AT%(4,5):Y=AT%(4,6)                                            '0.4 MG tank high & low
        AT%(7,11)=X:AT%(7,12)=Y                                            'Send to SS #1
        X=AT%(4,7):Y=AT%(4,8)                                              '3.7 MG tank high & low
        AT%(8,18)=X:AT%(8,19)=Y                                            'Send to booster #1
        AT%(9,18)=X:AT%(9,19)=Y                                            'And to booster #2
        return

MASTXFER5 X=AT%(5,5):Y=AT%(5,6)                                            '2.0 MG tank high & low
        AT%(6,11)=X:AT%(6,12)=Y:return                                     'Send to SS #3

USERIN PO%=0:get A$
        if len(A$)=0 then return                                           'Keyboard input?
        if DP<>0 goto USERIN1                                              'On main display?
        if A$="1" then gosub ACK:gosub ALRMRPT:goto USEROUT                'ACK key?
        if A$="2" then gosub SETUP:goto USEROUT                            'Setup?
        if A$="3" then gosub SETTIME:goto USEROUT                          'Set the clock
        if A$="4" then gosub PORT1:goto USEROUT                            'Port 1/string load?
        if A$="5" then gosub TTOGLDIAL                                     'Toggle dialer enable
        if A$="6" then DP=1:goto USEROUT                                   'Show AT%
        if A$="7" then DP=2:goto USEROUT                                   'Show AR%
        if A$="8" then gosub SETADDR                                       'Set unit address, com
USEROUT gosub CLRSCREEN:gosub DISPLAY:return
```

```
USERIN1 if A$="1" then DP=0:goto USEROUT                                 'Back to MAIN
        if A$="2" then gosub SETSTA:goto USEROUT                         'Set debug station address
        if A$="3" goto USERIN2                                           'Edit AT%() or AR%()
        if A$="5" then X=DB:gosub POLLTST:return                         'Remote tone test
        return
USERIN2 gosub BLANK7:print "Enter index to change...";
        gosub KBTOX
        if (XX<0 or X<1 or X>32) goto USEROUT                            'Watch range
        Y=X                                                              'Save index
        gosub BLANK7:print "Enter new value...";:gosub KBTOX
        if X<(-K3-1) or X>K3 goto USEROUT                                'Watch range
        if DP>1 then AR%(DB,Y)=X:goto USEROUT                            'Edit AR%()
        AT%(DB,Y)=X:PT=DB-1:goto USEROUT                                 'Edit AT%() & poll this sta

PORT1 gosub CLRSCREEN
        print "   *** STRING LOAD/PORT 1 CONTROL ***"
        print "Mode now...";
        if (peek(32527) and 2)=0 then print "ASCII":goto PORT1A
        print "MODBUS"
PORT1A print
        print "1 Enable MODBUS mode on port 1"
        print "2 Enable ASCII mode on port 1"
        print "3 Load speech config. strings."
        print
        print "Choose option (1-3)...";:gosub KBTOA
        if A$="1" goto SETADDR                                           'Enable MODBUS?
        if A$="2" goto SETASCII                                          'Enable ASCII?
        if A$="3" goto LLOADSTRG                                         'Load speech strings?
        return

SETASCII gosub CLRSCREEN
        print "Setting port 1 to ASCII..."
        gosub OPENRAM:X=3:gosub DELAYX
        X=64:Y=15:gosub VOTEXTOY:goto REBOOT                             'Disable MODBUS

LLOADSTRG gosub CLRSCREEN
        print "Send configuration now..."
        DT%(1)=120                                                       'Start timer
CONFIGIN1 PO%=1:gosub INSTRING                                           'Accept incoming config
        if DT%(1)=0 then PO%=0:print "TIMEOUT...":return
        DT%(1)=120                                                       'Start timer
        if left$(AA$,1)="\" then PO%=0:print "FINISHED, Bye":return      'Done?
        B$="":Y=len(AA$)
        if Y<4 goto CONFIGIN1                                            'Short or empty string?
        for I=1 to Y:A$=mid$(AA$,I,1)                                    'Get string #
        if A$<"0" or A$>"9" goto NUMDONE                                 'Numeric?
        B$=B$+A$:next
NUMDONE N=val(B$)                                                        'String number
        SN=int(N/HU):N=N-SN*HU                                           'SN=sta#, N=string #
        if len(AA$)<7 then Z$="":goto SAVSTRG                            'Blank string?
        Z$=right$(AA$,Y-I)                                               'Get string to save
SAVSTRG AS$(SN,N)=Z$:goto CONFIGIN1                                      'Save new string

INSTRING AA$=""                                                          'Get string from KB
```

```
INS1 get A$                                                          '& put in AA$
        if DT%(1)=0 then return                                      'Timed out?
        if len(A$)=0 goto INS1
        if A$=chr$(13) or A$=chr$(27) then print:return              'CR or ESC?
        if A$=chr$(8) goto BACKSPACE                                 'Backspace?
        print A$;:AA$=AA$+A$:goto INS1                               'Save, echo, go for more
BACKSPACE print A$;" ";A$;                                           'Blank old char
        if len(AA$)>0 then AA$=left$(AA$,len(AA$)-1)
        goto INS1

POLLTST AT%(X,4)=AT%(X,4) or G1%(15)                                 'Set bit for remote tone
        return

SETSTA gosub CLRSCREEN                                               'Set debug station address
        print:print:print "Enter new debug sta address..."
        gosub KBTOX
        if X<1 or X>20 then return
        DB=X:return

SETUP gosub CLRSCREEN                                                'Setup speech
        print "         *** SETUP ***"
        print "   1 Edit speech messages"
        print "   2 Edit speech strings"
        print "   3 Edit phone numbers"
        print "   4 Set rings to answer, now=";RG
        print "   5 Set callin security code"
        print "   6 Calibrate"
        print "Choose option (1-6)...";:gosub KBTOA
        if A$="1" then gosub TRAINSP:goto SETUP
        if A$="2" then gosub SPEDIT:goto SETUP
        if A$="3" then gosub EDPERSNUM:goto SETUP
        if A$="4" then gosub EDRINGS:goto SETUP
        if A$="5" then gosub EDSECUR:goto SETUP
        if A$="6" then gosub CALIN:goto SETUP
        return

EDSECUR gosub BLANK8:print "Enter new callin security code...";
        gosub KBTOX
        if XX<0 or X<0 then return
        SC=X:return                                                  'Save new security code

EDRINGS gosub BLANK8:gosub BLANK7                                    'Get rings to answer
        print "Enter rings to answer, 0 to 13..."
        print "Zero disables autoanswering.";:gosub KBTOX
        if XX<0 or X<0 or X>13 then return
        RG=X:return                                                  'Save rings to ans.

TTOGLDIAL DE=DE+1                                                    'Toggle dialer on/off
        if DE<0 or DE>2 then DE=0                                    'Watch range
        return

ACK AK=0:DO%(1)=0:return                                             'Stop speech, kill lamp

DISPLAY if DP>0 goto DISPL1                                          'Show display on LCD
```

```
        print "   *** COMMUNICATIONS CONTROLLER";DD;"***"
        print
        print "1 ACK           5 Dialer now...";
        if DE=0 then print "OFF  "                                  'Dialer enable?
        if DE=1 then print "LOCAL"
        if DE=2 then print "DIAL "
        print "2 Setup          6 Show AT%()"
        print "3 Set clock      7 Show AR%()"
        print "4 Port 1 control    8 Setup COM"
        print
        print "Choose option 1 to 8...";:return
DISPL1 if DP>1 goto DISPL2
        gosub CLRSCREEN:X=DB                                        'Debug display
        if (X<1 or X>20) then DB=3:X=3                              'Watch station range
        print "  *** AT%() DEBUG DISPLAY ***  Sta=";X
        for I=1 to 5:print I;AT%(X,I);"    ";                       'Column 1
        CC%=80:print I+5;AT%(X,I+5);"    ";                         'Column 2
        CC%=160:print I+10;AT%(X,I+10);"    "                       'Column 3
        next:goto MENU
DISPL2 gosub CLRSCREEN:X=DB                                         'Debug display
        if (X<1 or X>20) then DB=3:X=3                              'Watch station range
        print "  *** AR%() DEBUG DISPLAY ***  Sta=";X
        for I=1 to 5:print I;AR%(X,I);"    ";                       'Column 1
        CC%=80:print I+5;AR%(X,I+5);"    ";                         'Column 2
        CC%=160:print I+10;AR%(X,I+10);"    "                       'Column 3
        next
MENU CC%=0:CR%=64:print "1=DSP 2=Sta# 3=Set value";
        print " 5=TST";:return

POLL if DT%(7)<>0 then return    'Need to poll?
        DT%(7)=10                                                   'Short time for quick polls
POLL1 PT=PT+1                                                       'Next sta to poll
        if PT>9 then PT=4                                           'Done with stations?
        AT%(PT,1)=G1%(16)                                           'TX flag
        AT%(PT,0)=20:AR%(PT,0)=28                                   'TX, RX lengths
POLL2 PO%=5:print "E";:poke 542,PT:poke 541,96                      'Send poll
        gosub BLANK7:print "Polling station";PT;"attempt";11-CF%(PT);  'Show
        CF%(PT)=CF%(PT)-1                                           'Com fail counter
        if CF%(PT)=0 goto POLLFAIL                                  'Failed?
        if CF%(PT)<0 then CF%(PT)=-1:CN%(PT)=0                      'Limit range
        return
POLLFAIL CN%(PT)=0                                                  'Stop polling on fail
        X=PT:gosub CLRFLAGS                                         'Clear time sync and test flags on fail
        AR%(PT,5)=AR%(PT,5) or G1%(16):SN=PT:gosub STATTEST         'Fail flag on
        return

WATCHCOM X=0                                                        'Watch for incoming msgs
        for I=4 to 9                                                'Test for any msg
        if AR%(I,1)<0 then X=I:I=11
        next:PA=X
        if X=0 then return                                          'Had msg?
        AR%(PA,1)=0                                                 'Clr TLM flag
        CF%(PA)=10:CN%(PA)=0                                        'Restart com fail counter, clr need flag
        X=PA:gosub CLRFLAGS                                         'Clear clock sync & tone test flags
        if DP>1 and DB=PA then gosub DISPLAY                     'Show new data on LCD if AR%()
        gosub BLANK7:print "Received from station";PA
        if PA=4 or PA=5 then gosub XFER                          'Data transfers?
        if (AR%(PA,4) and G1%(15))<>0 then gosub SENDTONE        'Send test tone?
        SN=PA:gosub STATTEST                                     'See if any alarms
        return

CLRFLAGS AT%(X,4)=AT%(X,4) and G0%(14)                           'Clear clock sync flag
        AT%(X,4)=AT%(X,4) and G0%(15)                            'Clear tone command flag
        AT%(X,2)=AT%(X,2) and G0%(16)                            'Clear lockout reset flag
        return

SENDTONE DT%(1)=20                                               'Give 20 sec. test tone
        PO%=5:print "EX";                                        'Transmit tone on
        gosub BLANK7:print "Sending test tone to sta";PA
GOTMSLOOP if DT%(1)<>0 goto GOTMSLOOP                            'Wait til done
        gosub BLANK7:print "Done with test...";
        PO%=5:print "Y";:return

STATTEST X5=AR%(SN,5):X6=AR%(SN,6):I1=IM%(SN,1):I2=IM%(SN,2)
        if (SN>5 and SN<10) then X6=0:I2=0                       'Ignore 2nd word
        if X5=I1 and X6=I2 goto STATTEST2                        'Change?
STATTEST1 Y=0:for I=1 to 16                                      'Test if new one on
        if (X5 and G1%(I))<>0 and (I1 and G1%(I))=0 then Y=1
        if (X6 and G1%(I))<>0 and (I2 and G1%(I))=0 then Y=1
        if Y<>0 then I=17
        next
        if Y=0 goto STATTEST2                                    'Have any new ones?
        DO%(1)=1                                                 'New alarm lamp on
STATTEST3 if AK<>0 goto STATTEST2                                'Already have alarm?
        AK=1:PN=0                                                'Flag dialing, 1st PH#
        if CK%(6)=1 or CK%(6)=7 then DT%(4)=6900:goto STATTEST2  'Weekend?
        if CK%(2)<8 then DT%(4)=6900:goto STATTEST2              'Before 8 AM?
        if CK%(2)>16 then DT%(4)=6900:goto STATTEST2             'After 5 PM?
        if CK%(2)=16 and CK%(1)>29 then DT%(4)=6900:goto STATTEST2 '4:30 to 5?
        DT%(4)=7200                                              'Default 5 min.
STATTEST2 X=MF:for I=1 to 10                                     'Test if any alarms
        if I<6 then X=X+AR%(I,5)+AR%(I,6)                        'Both words for low sta.
        if I>5 then X=X+AR%(I,5)                                 'One word for high sta.
STATTEST4 next
        if X=0 then AK=0                                         'Kill dialing if none
        IM%(SN,1)=X5:IM%(SN,2)=X6:return                         'Save image

XFER Y=AR%(4,7):X=AR%(4,8)                                       'Reservoir levels
        AT%(8,10)=X:AT%(9,10)=X                                  '3.7 MG to boosters
        AT%(7,14)=Y                                              '0.4 MG to SS #1
        AT%(6,14)=AR%(5,7)                                       '2.0 MG to SS #3
        return

CLRFAIL AR%(PA,5)=AR%(PA,5) and G0%(16):CF%(PA)=10:return        'Clr com fail flag

WATCHDIAL if AK=0 or DE=0 then DL=2:return                       'See if need to annunciate
        if (AT%(1,4) and 1)=0 goto WATCHD1                       'Master ACK?
        AT%(1,4)=AT%(1,4) and G0%(1):gosub ACK:return            'Master ACK...clr it
```

```
WATCHD1 gosub ANYALARMS                                              'Any alarms still present?
        if AF=0 then AK=0:return                                     'Do ACK if none
        if DT%(4)=0 then DT%(4)=7200:AK=1:PN=0                       'Restart timer, 1st operator
        if (DT%(4)>6900 or AK=2) goto LOCAL                          'Local?
DIALNOW if DE<2 goto LOCAL                                           'Enable dialer?
        if DT%(8)<>0 then return                                     'Waiting to dial next operator?
        gosub DIALOPER                                               'Dial one operator
        if (PN>7 and AK<>0) then AK=2                                'Finished list?
        return
LOCAL if DL=2 then gosub SPKLOCAL:DL=1                               'Spk alarm immediately?
        if (CK%(1) and 1)=0 and DL=0 then DL=1:gosub SPKLOCAL        'Local @ 2 min.
        if (CK%(1) and 1)<>0 then DL=0                               'Reset annun flag
        return

ANYALARMS X=0                                                        'Test if any alarms
        for SN=2 to 9
        if SN<6 then X=X+abs(AR%(SN,5))+abs(AR%(SN,6))
        if SN>5 then X=X+abs(AR%(SN,5))
ANYAL1 next
        AF=X:return                                                  'Flag<>0=alarm

DIALOPER PN=PN+1                                                     'Next PH #
        if PN>8 then PN=8:return                                     'Stop after 8 numbers
        if len(PH$(PN))=0 goto DIALOPER                              'Have ph numbr?
        gosub DIALOUT                                                'Dial PH #
        X=10:gosub DELAYX                                            'Wait
        X=len(CO$(PN))                                               'Security code?
        if X<1 goto DIALVERB                                         'Assume verbal if no code
        A$=left$(CO$(PN),1)                                          'Get 1st char of code
        if (A$>="0" and A$<="9") goto DIALVERB                       '1st char numeric==>verbal
        PO%=5:print "A";                                             'Modem 2-wire for pager code
        for I=1 to X-1:A$=mid$(CO$(PN),I+1,1)                        'Get 1 char from code
        print A$;                                                    'Send char to pager
        next
        X=3:gosub DELAYX:DT%(8)=600                            'Wait 3 sec code, 10 min callback *
        gosub HANGUP                                                 'Hangup *
        gosub BLANK7:print "Done dialing.":return'Tell user *
DIALVERB PO%=5:print "E"                                             'Speech to 2-wire channel
        gosub REPORTOUT                                              'Report then get security for ACK
        gosub HANGUP                                                 'Hang up phone
        gosub BLANK7:print "Done dialing.":DT%(8)=60:return          '60 sec callback time

REPORTOUT RR=4:gosub CLRTTONES                                       'Up to 4 repetitions
REPAGAIN PO%=4:print "P13";:gosub SPKWAIT                            '"This is the...autodialer "
        gosub VERBOUT                                                'Report alarms
        if RR=-1 then gosub ACK:gosub GOODBYE:return                 'Had local ack?
        RR=RR-1
        'Decr repeat counter
        if RR>0 then PO%=4:print "P19P20";:goto REPAGAIN             'Repeat?
NOWSECUR gosub SECURITY:gosub GOODBYE:return                         'Ask for & get security

SPKLOCAL gosub CLRTTONES                                             'Clear touch tones
        RR=1:gosub VERBOUT                                           'Annunciate alarms locally
        if RR=-1 then ACK=0:return                                   'Had local ACK?
        PO%=4:print "P20";:gosub SPKWAIT:gosub VERBOUT          '"Repeating..."
        if RR=-1 then AK=0                                                  'Had local ACK?
        return

VERBOUT if MF=0 goto VERB11                                     'State all active alarms
        PO%=4:print "P201P46";:gosub SPKWAIT                    '"Master has failed"
        PO%=0:get A$                                            'Master ACK?
        if len(A$)<>0 then RR=-1
VERB11 for SN=2 to 15                                           'State all active alarms
        if (SN>5 and AR%(SN,5)=0) goto VERBNXT                  'One word for hi sta. *
        if (SN<6 and (AR%(SN,5)+AR%(SN,6))=0) goto VERBNXT      'Tst if sta has alrm *
        PO%=4:print "P";SN+200;:gosub SPKWAIT                   'Identify station
        for BT=1 to 16                                          'Scan 1st 16 alarms
        PO%=0:get A$                                            'KB ack?
        if len(A$)=0 goto VERBOUT2
        if A$="1" then RR=-1
VERBOUT2 PO%=5:get A$                                           'Touchtone char?
        if ((A$>="0" and A$<="9") or A$="*" or A$="#") then RR=-2   'ACK?
        if (AR%(SN,5) and G1%(BT))=0 goto VERBBIT               'Alarm?
        PO%=4:print AS$(SN,BT);:gosub SPKWAIT                   'Report alarm
VERBBIT if RR<0 then BT=HU                                      'Had ACK?
        next                                                                'Next bi
        if BT=HU goto VERBNXT
        if SN=6 or SN=7 goto VERBNXT                            'Skip 2nd word on SS1 & SS3
        for BT=1 to 16                                          'Scan 2nd 16 alarms
        PO%=0:get A$                                            'KB ack?
        if len(A$)=0 goto VERBOUT3
        if A$="1" then RR=-1
VERBOUT3 PO%=5:get A$                                           'Touchtone char?
        if ((A$>="0" and A$<="9") or A$="*" or A$="#") then RR=-2   'ACK?
        if (AR%(SN,6) and G1%(BT))=0 goto VERBBIT3              'Alarm?
        PO%=4:print AS$(SN,BT+16);:gosub SPKWAIT                'Report alarm
VERBBIT3 if RR<0 then BT=HU                                     'Had ACK?
        next                                                                'Next bi
VERBNXT if RR<0 then SN=HU                                      'Had ACK?
        next                                                               'Next
station
VERBEX return

GOODBYE PO%=4:print "P18";:gosub SPKWAIT:return                 'Goodbye

SECURITY TY=3                                                   'Get security code...3 tries
SECUR2 DT%(1)=20:PO%=5:AA$=""                                   'Get security code
        X=len(CO$(PN))                                          'Test security code
        if X=0 then return                                      'No code necessary
        gosub CLRTTONES:PO%=4:print "P14";:gosub SPKWAIT        '"Pls enter sec. code"
SECUR1 if DT%(1)=0 goto MISMATCH                                'Out of time, no security
        PO%=5:get A$
        if A$="*" goto TEST                                     'End of code?
        if A$>="0" and A$<="9" then AA$=AA$+A$                  'Number?
        goto SECUR1
        if X<>len(AA$) goto MISMATCH                            'Lengths match?
TEST for I=1 to X
        if mid$(AA$,I,1)<>mid$(CO$(PN),I,1) goto MISMATCH       'Mismatch?
        next
MATCH AK=0:PO%=4:print "P15P16";:gosub SPKWAIT:return                    'Match

MISMATCH PO%=4:print "P15P17";:gosub SPKWAIT                             'Mismatch..rejected
        TY=TY-1
        if TY<=0 then return                                             'Expended tries?
        goto SECUR2                                                      'Go try again

CLRTTONES PO%=5:for I=1 to 20:get A$:next:return                         'Clr touch tones

SPKWAIT PO%=4:print                                          'Send carriage return to start synthesizer
SPKAGAIN PO%=4:get A$                                        'Get status of synthesizer
        if A$="I" then return                                'Wait for "I" (idle) indication
        goto SPKAGAIN

HANGUP PO%=5:print "K"                                       'Hang up phone
        return

OFFHOOK PO%=5:print "L";:return                              'Pick up phone

WIRE2 PO%=5:print "A";:return

DIALOUT if len(PH$(PN))=0 then return                                    'Dial PH #(PN)
        gosub OFFHOOK                                                    'Off hook
        gosub BLANK7:print "Dialing operator at ";PH$(PN);"...";
        X=3:gosub DELAYX                                                 'Wait 3 sec.
        PO%=5:print "A";                                                 'Set modem to 2-wire for dialing
        for I=1 to len(PH$(PN)):A$=mid$(PH$(PN),I,1)                     'Get 1 char from phone #
        if A$="-" then X=2:gosub DELAYX:goto NXTCH                       'Delay 2 sec. if "-"
        print A$;                                                        'Otherwise send char to dialer
NXTCH next
        X=3:gosub DELAYX:return                                          'Wait 3 sec for dialing

BLANK7 PO%=0:CC%=0:CR%=56                                                'Blank LCD line 7
        print "                              ";
        CC%=0:return

BLANK8 PO%=0:CC%=0:CR%=64                                                'Blank LCD line 8
        print "                              ";
        CC%=0:return

DELAYX DT%(1)=X                                                          'Delay X seconds
DELAY1 if DT%(1)<>0 goto DELAY1
        return

CLRSCREEN PO%=0:print chr$(12):CC%=0:CR%=8:return

SETADDR V1=32512:V2=V1+16:V3=V2+16                                       'Start of voted table
        gosub CLRSCREEN
        gosub OPENRAM:X=3:gosub DELAYX                                   'Open RAM & wait for write protect
        gosub ADDRESS                                                    'Set address
        gosub SETCRCOPS                                                  'Setup for CRC radio TLM
REBOOT print "REBOOTING...PLEASE WAIT":X=2:gosub DELAYX
        poke 4,3:poke 5,136:X=USR(0)                                     'Jmp to $8803 (MAIN)
```

```
DELAYX DT%(1)=X                                                    'Delay X seconds
DELAY1 if DT%(1)<>0 goto DELAY1
        return

ADDRESS X=1:print:print "Setting address to";X
        poke V1+3,X:poke V2+3,X:poke V3+3,X:return                 'Store new unit address

VOTEXTOY poke V1+Y,X:poke V2+Y,X:poke V3+Y,X:return                'Poke X to VOTDAT,Y...

OPENRAM print "Opening RAM..."
        X=peek(969) and 191:poke 969,X:poke 34800,X:return         'Open RAM

SETCRCOPS print "Setting up CRC flags & com delays..."
        X=1:Y=1:gosub VOTEXTOY                                     'Setup RAM for CRC, radio..parity
        X=22:Y=2:gosub VOTEXTOY                                    'Baud, stop bits, word length
        X=63:Y=10:gosub VOTEXTOY                                   'Low tones...15 rings
        X=9:Y=11:gosub VOTEXTOY                                    'CRC communications
        X=42:Y=12:gosub VOTEXTOY                                   'Delay=.7 sec before XMIT
        X=66:Y=15:gosub VOTEXTOY                                   'Setup modem connection & MODBUS
        return

SPEDIT gosub CLRSCREEN                                             'Edit speech strings
        print:print:print "Choose station # (1-10)..."
        gosub KBTOX
        if X<1 or X>10 then return                                 'Watch range
        SN=X
SPEDITAL X=0:gosub SHOWSPST                                        'Show speech ID strings
        gosub BLANK8:print "Choose alarm to change (1-16)...";
        gosub KBTOX
        if X<1 or X>16 goto SPEDIT2                                'In range?
        Y=X:AS$(SN,Y)=""                                           'Blank old string
SPEDIT1 X=0:gosub SHOWSPST                                         'Show strings now
        gosub BLANK8:print "Enter # of next phrase (1-999)...";
        gosub KBTOX
        if XX<0 or X<1 or X>999 goto SPEDITAL                      'In range?
        A$=str$(X):X=len(A$)-1:A$=mid$(A$,2,X)                     'Get phrase ID
        AS$(SN,Y)=AS$(SN,Y)+"P"+A$                                 'Save in speech cmd format
        PO%=4:print AS$(SN,Y)                                      'Say it
        goto SPEDIT1
SPEDIT2 X=16:gosub SHOWSPST                                        'Show speech ID strings
        gosub BLANK8:print "Choose alarm to change (17-32)...";
        gosub KBTOX
        if X<17 or X>32 then return                                'In range?
        Y=X:AS$(SN,Y)=""                                           'Blank old string
SPEDIT3 X=16:gosub SHOWSPST                                        'Show strings now
        gosub BLANK8:print "Enter # of next phrase (1-999)...";
        gosub KBTOX
        if XX<0 or X<1 or X>999 goto SPEDIT2                       'In range?
        A$=str$(X):X=len(A$)-1:A$=mid$(A$,2,X)                     'Get phrase ID
        AS$(SN,Y)=AS$(SN,Y)+"P"+A$                                 'Save in speech cmd format
        PO%=4:print AS$(SN,Y)                                      'Say it
        goto SPEDIT3
```

```
SHOWSPST gosub CLRSCREEN:print "        *** ALARM STRINGS ***"
         for I=1 to 6
         print I+X;AS$(SN,I+X);                                    'Show strings
         CC%=80:print I+6+X;AS$(SN,I+6+X);                         'Col 2
         if I>4 then print:goto SHOWSPST1
         CC%=160:print I+12+X;AS$(SN,I+12+X)                       'Col 3
SHOWSPST1 next:return

EDPERSNUM gosub CLRSCREEN
         print "        *** PERSONNEL NUMBER LIST ***":print
         print "  NUMBER    CODE     NUMBER    CODE"
         for I=1 to 4:CC%=0:print I;PH$(I);
         CC%=84:print CO$(I);:CC%=128:print I+4;PH$(I+4);
         CC%=212:print CO$(I+4):next
         print "Choose number to change (1-8)...";:gosub KBTOA
         if A$<"1" or A$>"8" then return
         I=val(A$):gosub BLANK8:print "Enter new phone # for entry #";I;
         gosub INSTRING:PH$(I)=AA$
         gosub BLANK8:print "Enter new security code for entry #";I;
         gosub INSTRING:X=len(AA$)
         if X<1 goto EDPERSNUM                                     'Have code?
         A$=left$(AA$,1)                                           'Get 1st char
         if (A$>="0" and A$<="9") goto EDPERS1                     'Numeric 1st char?
         A$=right$(AA$,X-1)                                        'Get all but 1st char
         AA$="P"+A$+"#"                                            'Put "P" on front, # on back
EDPERS1 CO$(I)=AA$:goto EDPERSNUM                                  'Save security code

INSTRING AA$=""                                                    'Get string from KB
INS1 get A$                                                        '& put in AA$
         if len(A$)=0 goto INS1
         if A$=chr$(13) or A$=chr$(27) then print:return           'CR or ESC?
         if A$=chr$(8) goto BACKSPACE                              'Backspace?
         print A$;:AA$=AA$+A$:goto INS1                            'Save, echo, go for more
BACKSPACE print A$;" ";A$;                                         'Blank old char
         if len(AA$)>0 then AA$=left$(AA$,len(AA$)-1)
         goto INS1

TRAINSP PX%=0:PO%=PX%:gosub CLRSCREEN
         print "*** SPEECH EDITOR ***"                             'Enable user to train speech
         print
         print "1=Record"
         print "2=Playback"
         print "3=Delete"
         print
         print "Enter choice..."
         DT%(1)=300:gosub KEYSTRK                                  'get user choice.
         if X<1 or X>3 then return                                 'terminated.
         K=1
         'initialize message # to
         on X gosub RECORD,PLAYBACK,DELETE                         '1, branch to choice.
         goto TRAINSP

RECORD print "Ready to record message #";K;"."
         print "Or enter different number (1 to 999)."
```

```
          print "Recording starts and stops with ENTER key: ";
          XZ=K:gosub MSGNUM
          if K=0 then return                                    'exit if invalid.
          if XZ<>K goto RECORD                                  'Reprompt if new msg #
          CM$="R"+str$(K):gosub SPEECH                          'record command.
          if UM$="" goto GOODREC                                'command accepted.
          print UM$:goto RECAGAIN                               'Problem, show user message.
GOODREC  print:print "Recording message #";K;" ...."
          DT%(1)=300                                            'give user more than max
LOOKSTOP gosub KEYSTRK                                          'recording time.
          if len(A$)=0 then return                              'user didn't stop.
          if asc(A$)<>13 goto LOOKSTOP                          'no [ENTER] key yet.
'User stops speech process:
          PO%=PX%:CM$="C":gosub SPEECH:print UM$                'Cancel recording
'Above: Show recording time left
          PO%=4:print "P"+str$(K)                               'Say message
RECAGAIN K=K+1:PO%=PX%:print:goto RECORD                        'auto advance to next msg.

PLAYBACK print "Ready to playback message #";K;"."
          print "Or enter different number (1 to 999)."
          print "Playback starts with [ENTER] key: ";
          gosub MSGNUM
          if K=0 then return                                    'exit if invalid msg #.
          PO%=5:print "EL";                                     'Send to phone also
          CM$="P"+str$(K):gosub SPEECH                          'command to speech board.
          if UM$="" goto PLAYING
          print UM$:goto PLAYAGIN
PLAYING  print:print "Playing message #";K;" ...."
MOREPLAY gosub SPEECH                                           'look for end of playback.
          if UM$="" goto MOREPLAY                               'still busy.
          print UM$
PLAYAGIN K=K+1:print:goto PLAYBACK                              'auto advance msg #, send

DELETE print:print "1=Delete single message"
          print "2=Delete range"
          print "Enter option (1-2)...";:gosub KBTOA
          if A$="1" goto DELETE1
          if A$="2" goto DELRANGE
          return

DELRANGE print:print "Enter lowest message to delete...";:gosub KBTOX
          if XX<0 or X<1 or X>999 then return
          LS=X                                                          'Lowest
          print:print "Enter highest message to delete...";:gosub KBTOX
          if XX<0 or X<1 or X>999 or X<LS then return
          MS=X                                                          'Highes
          for K=MS to LS step -1:gosub DELETEK:next:return

DELETE1 print:print "Enter message # to delete (1-999)";:gosub KBTOX
          if XX<0 or X<1 or X>999 then return
          K=X:gosub DELETEK:goto DELETE1                        'Delete it

DELETEK print "MSG=";K
          CM$="D"+str$(K):gosub SPEECH                          'command to speech board.
```

```
        if UM$="" goto DELTING
        print UM$:return
DELTING print:print "Deleting message #";K;" ...."
PACKING gosub SPEECH                                              'watch for completion.
        if UM$="" goto PACKING                                    'still busy.
        print UM$:return                                          'Result.

MSGNUM DT%(1)=60:gosub KEYBOARD
        if AA$=chr$(13) then return                               'only an [ENTER].
        K=int(XX)
        if K<1 or K>1000 then K=0                                 'out of range.
        return

SPEECH PO%=4:UM$=""                                               'speech port.
        if CM$="C" then print CM$;:CM$=""                         'no line feed on cancel.
        if CM$<>"" then print CM$:CM$=""                          'any other command.
FINEX get A$                                                      'read speech response.
        if A$="W" or A$="I" goto TRANSMSG                         'char interchange done.
        if A$="A" goto FINEX                                      'acknowledge useless.
        UM$=UM$+A$:goto FINEX                                     'save E, M and #.
TRANSMSG PO%=PX%                                                  'ready for LCD msg.
        if UM$="" then return                                     'no msg at this time.
        if left$(UM$,1)="M" goto TIMELEFT                         'memory remaining msg.
        N=val(right$(UM$,1))                                      'error message, look
        on N goto ERM1,ERM2,ERM3,ERM4,ERM5,ERM6
        UM$="":return
TIMELEFT UM$=mid$(UM$,2):MR=val(UM$)                              '# of Kbytes of RAM left.
        TL=int(MR*26.5)/100                                       'convert to seconds.
        UM$="Recording time remaining (sec.)="+str$(TL)  'put into msg.
        return

ERM1 UM$="Out of voice memory.":return
ERM2 UM$="Message already exists for this message #.":return
ERM3 UM$="No message exists for this message #.":return
ERM4 UM$="No message number was specified.":return
ERM5 UM$="Too many characters in message number.":return
ERM6 UM$="Playback message que is full.":return

        'KEYBOARD checks the keyboard for user entry.
        'For single keystroke enter at KEYSTRK.

KEYBOARD PO%=PX%:AA$="":CF%=0:XX=-1                               'initialize.
ANOTHER gosub KEYSTRK                                             'get next key.
        if len(A$)=0 then AA$="":XX=-1:print:return               'no entry.
        if asc(A$)=13 then AA$=AA$+A$:print:return                '[enter].
        if X<>-1 goto ADDCHAR                                     'number.
        if A$="-" and CF%=0 goto ADDCHAR                          'leading minus.
        if A$<>"." or CF%=2 goto KEYWAIT                          'allow one dec
        CF%=2                                                     'point.
ADDCHAR print A$;:AA$=AA$+A$:XX=val(AA$)                          'display char
        if CF%=0 then CF%=1                                       '& add to str.
KEYWAIT DT%(1)=120:goto ANOTHER                                   'wait again.
KEYSTRK PO%=PX%:X=-1:get A$                                       'new key?
        if len(A$)<>0 goto XVALUE                                 'evaluate it.
        if DT%(1)>0 goto KEYSTRK                                        'still waiting.
        return                                                          'time out.
XVALUE if A$>="0" and A$<="9" then X=val(A$)                            'got a digit.
        return

BINMASK X=1  'Setup mask fields
        for I=1 to 15:G1%(I)=X:G0%(I)=-X-1:X=X*2:next
        G1%(16)=-32768:G0%(16)=32767:return

'KBTOX reads keyboard and puts value in X
'Allows 120 sec for keystroke
'At end, if error or CR only, XX=-1, otherwise X has value entered

KBTOX J=CC%:DT%(1)=120:X=0:XX=0:I=0:L=0:B$=""                           'Get KB value to X
KBTOX1 if DT%(1)=0 then XX=-1:return
        get A$                                                          'Get character
        if len(A$)=0 goto KBTOX1
        DT%(1)=120                                                      'Restart timer on keystroke
        if A$="C" or A$="D" or A$="U" goto CLRIT
        if X<0 and A$="-" goto CLRIT                                    '2 "-" signs?
        if L<>0 and A$="." goto CLRIT                                   '2 decimals?
        I=I+1:print A$;                                                 'Count chars received & echo
        if A$="-" then X=-1                                             'Flag "-"
        if I=1 and A$=chr$(13) then XX=-1:return                        'CR only?
        if A$=chr$(13) then X=val(B$):return
        B$=B$+A$:goto KBTOX1                                            'Save char & go for more
CLRIT CC%=J:for K=1 to I:print " ";:next:CC%=J:goto KBTOX               'Clr entry

KBTOA DT%(1)=120                                                        'Wait up to 120 sec. for keystroke
KB1 if DT%(1)=0 then A$="C":return                                      'Timeout?
        get A$
        if len(A$)=0 goto KB1
        return                                                          'Return with keystroke in A$

KBTOXTT gosub CLRTTONES                                                 'Get KB value to X (touchtone)
        J=CC%:X=0:XX=0:I=0:L=0:B$=""
KBTOXTT1 gosub KBTOATT                                                  'Get 1 char.
        if A$="C" then XX=-1:return                                     'Timed out?
        if A$="#" then A$="."                                           'Use # as dp
        if L<>0 and A$="." goto CLRITT                                  '2 decimals?
        I=I+1                                                           'Count char rcvd
        if I=1 and A$="*" then X=-1:XX=-2:return                        'CR only?
        if A$="*" then X=val(B$):return
        B$=B$+A$:goto KBTOXTT1                                          'Save char & go for more
CLRITT goto KBTOXTT                                                     'Clr entry

KBTOATT DT%(1)=20                                                       'Wait for touchtone keystroke
        gosub CLRTTONES                                                 'First clear tones
KB1T if DT%(1)=0 then A$="C":return                                     'Timeout?
        PO%=5:get A$
        if A$>="0" and A$<="9" then return                              'Have char?
        if A$="*" or A$="#" then return                                 'Ditto
        goto KB1T
```

```
SETTIME gosub TIMENOW:print "Enter new day of week:"
        print "1=Sun, 2=Mon, 3=Tue, 4=Wed"
        print "5=Thu, 6=Fri, 7=Sat";:gosub KBTOA
        if A$<"1" or A$>"7" goto MONTHIN
        CK%(6)=val(A$)
MONTHIN gosub TIMENOW:print "Enter new month (1-12):":input I
        if I<1 or I>12 goto DAYOFMONTH
        CK%(4)=I
DAYOFMONTH gosub TIMENOW:print "Enter day of month (1-31):":input I
        if I<1 or I>31 goto YEARIN
        CK%(3)=I
YEARIN gosub TIMENOW:print "Enter new year (0-99):":input I
        if I<0 or I>99 goto HOURIN
        CK%(5)=I
HOURIN gosub TIMENOW:print "Enter new hour (0-23):":input I
        if I<0 or I>23 goto MINUTEIN
        CK%(2)=I
MINUTEIN gosub TIMENOW:print "Enter new min. (0-59):":input I
        if I<0 or I>59 then return
        CK%(1)=I:CK%(0)=0:return

TIMENOW gosub CLRSCREEN:gosub RTC
        print "Present date/time:";JJ$:return

'**********Setup RTC string in JJ$:**********

RTC I=CK%(6)
 if I<1 or I>7 then JJ$="    ":goto SKIPDAY
 I=3*(CK%(6)-1)+1:C0=CK%(1)
 JJ$=mid$("SUNMONTUEWEDTHUFRISAT",I,3)+" "                'Day of week
SKIPDAY I=4:gosub TIMEX :JJ$=JJ$+"/"                      'Month
 I=3:gosub TIMEX :JJ$=JJ$+"/"                             'Day of month
 I=5:gosub TIMEX :JJ$=JJ$+" "                             'Year
 I=2:gosub TIMEX :JJ$=JJ$+chr$(58)                        'Hours:
 I=1:gosub TIMEX :JJ$=JJ$+" ":return                      'Minutes
TIMEX A$=str$(CK%(I))                                     'read clock & format to 2 chars.
 IF len(A$)<3 then JJ$=JJ$+"0"+right$(A$,1):return
 JJ$=JJ$+right$(A$,2):return

'CALMAN calibration routines
'Interacts with user to gather coefficients for 2 point calibration

'For analog input calibration:
'       M and B are stored in AM() and AB() respectively
'       To avoid losing cal when editing program, the calibration
'       constants are stored in the reserved area from
'       $1F00 through $1FFF.  These are loaded by CALINIT upon
'       boot up.  A total of 32 analog channels can be stored in this
'       area.  This routine allocates space for 24 analog inputs, and
'       8 analog outputs.  They are arranged thus:
'               $1F00-$1F03=AM(1)*65536
'               $1F04-$1F07=AB(1)*65536
'               $1F08-$1F0B=AM(2)*65536
'
```

```
'
'                    $1FC0-$1FC3=OM(1)*65536
'                    $1FC4-$1FC7=OB(1)*65536

'In realtime operation, result would be:
'           result(I)=AM(I)*AI%(I)+AB(I) for inputs 1 to 11
'           result(I)=AM(I)/AI%(I)+AB(I) for inputs 12 and up.

'For analog output calibration:
'           M and B are stored in OM() and OB() respectively

'In realtime operation, result would be:
'           AO%(I)=OM(I)*value(I)+OB(I)

'CALINIT reads reserved user area and gathers saved cal coefficients.
'           Should be executed at beginning of program.

CALINIT for I=1 to 11
        II=(I-1)*8:gosub CALRAMINX                          'Get AM() cal value to X
        AM(I)=X/65536                                       'Scale down and save
        II=(I-1)*8+4:gosub CALRAMINX                        'Get AB() cal value to X
        AB(I)=X/65536                                       'Scale down & save
        next
        return

CALRAMINX J=7936+II                                         'Get X from reserved area, byte J
        X=peek(J)+256*peek(J+1)+65536*peek(J+2)
        XX=peek(J+3) and 128                                'Get sign
        II=peek(J+3) and 127                                'Get MS byte, no sign
        X=X+16777216*(II)
        if XX=0 then return
        X=-X:return

CALRAMOTX J=7936+II:I1%=0                                   'Save X to location in reserved area, byte J
        if X<0 then I1%=128:X=-X                            'Get sign of X
        X=X*65536:XX=int(X/16777216)
        X1%=XX and 127 or I1%:poke J+3,X1%                  'MS byte (1)
        X=X-XX*16777216:X1%=int(X/65536):poke J+2,X1%       'Byte (2)*****
        X=X-X1%*65536:X1%=int(X/256):poke J+1,X1%           'Byte (3)*****
        X=X-X1%*256:poke J,int(X):return                    'LS byte (4)

CALIN gosub CLRSCREEN
        print "Enter input channel to calibrate."
        print "Allowed range is 1 through 11, 0 exits":input X
        if X<1 or X>11 goto CALDONE
        CH=int(X)                                           'Save channel we're calibrating
        print "Apply known low value to analog input";CH
        print "Key in corresponding engrg. units value.":input X
        A1=X:A2=AI%(CH)                                     'Save it and A/D output
        print "Apply known high value to analog input";CH
        print "Key in corresponding engrg. units value.":input X
        A3=X:A4=AI%(CH)                                     'Save it and A/D output
        if CH>11 then A2=1/A2:A4=1/A4                       '1/AI%(I) if expansion A/D
        if abs(A2-A4)>.000001 goto COMPUT
        print "Input value error...restarting cal."
        for I=1 to 1000:next                                      'Delay
        goto CALIN
COMPUT M=(A1-A3)/(A2-A4)                                          'Compute M
        B=A1-M*A2                                                 'and B

'Now show result of this calibration
        gosub CLRSCREEN                                           'Clear the LCD
        A2=100:A3=0.5                                             'Use for significance control
CALOOP CR%=8:CC%=0                                                'Set display cursor location
        print "Present AI";CH;"=";AI%(CH);"  "                    'Show user A/D input
        if CH<12 then A1=int((M*AI%(CH)+B)*A2+A3)/A2              'Control significance
        if CH>11 then A1=int((M/AI%(CH)+B)*A2+A3)/A2              'Expansion AI's
        print "Engrg. value=";A1;"  "                             'Show resulting value
        print "Key in 1 to save"
        print "Key in 2 to discard"
CAL1 get A$
        if len(A$)=0 goto CALOOP                          'Watch for keystroke
        if A$<>"1" goto CALIN
        AM(CH)=M:AB(CH)=B                                 'Save coefficients
        X=AM(CH):II=(CH-1)*8:gosub CALRAMOTX              'Put in reserved area
        X=AB(CH):II=(CH-1)*8+4:gosub CALRAMOTX            'Ditto
        goto CALIN                                        'Return for more

CALDONE print "Exiting calibration routines."
        for I=1 to 1000:next:return                       'Delay & exit
```

INDEX

INDEX

INDEX

RUGID COMPUTER
6305 Elizan Dr. NW
Olympia, WA 98502

360.866.4492 FAX 360.866.8074

Document 392080