Multipro \ Dualpro

PROGRAMMING & BASIC LANGUAGE HANDBOOK

Marathon Monitors Inc.

This manual has part number F200043

Revision Date 11 01 97(First Release) Revision number 0
Revision 1 Date 05 19 98(Add Recipe editing) Revision number 1.0
Revision 2 Date 07 05 06
Revision 3 Date 25 July 08 (added page numbers and update address)

For assistance please contact:

Marathon Monitors Inc.

TEL: +1 513 772 1000 • FAX: +1 513 326 7090

Toll-Free North America +1-800-547-1055

erika.leeds@group-upc.com

Marathon Monitors Inc.

TABLE OF CONTENTS:

Multipro Programming:

The Multipro employs a Dual Programmer. A custom background program will normally always run in the background. This custom background program is written based on the system design and handles the following functions:

a. Data logging
b. Alarm handling
c. Communications with various equipment
d. Deviation alarms
e. Probe maintenance
f. Auto cal functions (if any)
g. Event handling
h. Recipe management

The background program is written in MSI Logic language. The Logic language is powerful and flexible and allows the MSI Multipro to accomplish a wide variety of operations and interface to a wide range of equipment. However, the complexity of the Logic language limits its usefulness as a "recipe input" language.

Many MSI customers are familiar with recipe entry using the MSI Recipe language. Previously, this recipe language was available only on MSI v3 and 3.5 (including MultiCarb) instruments. The recipe language provides a simple method for entering recipes.

The Multipro provides the capability of using either Logic language or Recipe programs in the foreground. Using a custom Logic language program in the background and recipes in the foreground provides the full flexibility and complexity of a Multipro. This maintains the ease of recipe entry associated with the v3 and 3.5 MSI instruments.

Recipe Programming

The Recipe language uses a STEP/OPCODE approach to recipe design. The STEP approach is very similar to what an operator would do if he were manually controlling the process.

The following are the OPCODEs used in the recipe language. Theses will be followed by sample recipes, which can be run to test various functions of the Multipro.
Recipe Language operations codes:

OPCODE	MNEMONIC	DATA RANGE	DESCRIPTION
NOP	NOP		No operation
A	ALARM	1 to 79	Sound and display an alarm to summon and inform the operator of a message. XXXX is the programmed message number.
B	BRANCH	0 to 19.19	Instructs a program where to go. After an inquiry: XX.XX = TT.FF. 0000 branches out of the program.(TT= inquire True, FF= inquiry False)
C	CARB SETP	-100 to 2030	Changes the carbon setpoint to the value XXXX. See notes
c	CARB TEST	-100 to	Check if Dewpoint or %C

OPCODE	MNEMONIC	DATA RANGE	DESCRIPTION
		2000	is above the specified value. see notes.
d	ADD REFN	-128 to 127	add to the reference number.
E	EVENT	000.0 to 063.0 000.1 to 063.1	Turns an output ON or OFF or waits for an input condition. The Programmer Waits for an acknowledgment that the change has occurred before advancing to the next step. The farthest right DATA character indicates ON (.1) or OFF (.0) state. The first three DATA characters indicate which INPUT or OUTPUT is affected.

OPCODE	MNEMONIC	DATA RANGE	DESCRIPTION
F	FLAG	000.0 to 047.0	Turns a Flag ON or OFF. Flags are used to convey information to the background program. The Programmer does not wait after setting a flag (the LIMIT OPCODE has no effect.
		000.1 to 047.1	The farthest right DATA character indicates ON (.1) or OFF (.0) state. The first three DATA characters indicate which FLAG is affected. The Background program would access these flags as bits 4 through 15 of 00 for Flags 0 - 11 respectively.

OPCODE	MNEMONIC	DATA RANGE	DESCRIPTION
G	GOSUB	0000 to 0201	Allows one program to execute another program and then continue. Any program can be called as a subroutine as long as it does not call a subroutine. When a subroutine ends, the calling program is reloaded and restarted at the step following the G OPCODE. See notes A GOSUB 201 will cause a program to be called whose number is equal to the REFERENCE NUMBER.
H	TEMP SETP	0000 to 4000	Changes the temperature controller SETPOINT. NOTE 2

Recipe Language operations codes:

OPCODE	MNEMONIC	DATA RANGE	DESCRIPTION
h	TEMP TEST	0 to 4000	Check to see if the temperature is above the specified value.
I	DELAY	2 to 250	Insert a delay in real-time seconds.
J	JUMP	0 to 201	Jump to another program and continue executing the new program. The programmer never returns to the program with a J OPCODE unless called by a G OPCODE. A JUMP 0000 will reload and execute the currently running program. A JUMP 201 will cause a jump to The program whose number is equal to the reference number.

OPCODE	MNEMONIC	DATA RANGE	DESCRIPTION
L	LIMIT	5 to 4000	Limit the amount of time a programmer should wait for something to happen before sounding an alarm. The L OPCODE only works when used with another OPCODE. The data in a LIMIT statement may be used to indicate temperature. see notes.
m	IN A TEST	0 to 4000	Check the temperature from input A.
n	SET REFN	0 to 255	Set a new reference number.
o	IN B TEST	0 to 4000	Check the O2 mV from input B.
P	SET PROF	0 to 999	Allow the Process Factor to be altered by the programmer. See note 3.
q	REFN TEST	0 to 4000	Check to see if the reference number is above the specified value.

Recipe Language operations codes:

OPCODE	MNEMONIC	DATA RANGE	DESCRIPTION
r	RAMP	5 to 4000	Specifies the RAMP time from current temperature SETPOINT to the new SETPOINT. Must be followed by an H OPCODE. xx.xx is the RAMP time in hours and minutes.
S	SOAK	5 to 4000	Hold at heat for a specified time. Data entered is the soak time in hours and minutes.
T	MTMR SET	0 to 4000	set the master timer see note 4.
t	MTMR TEST	0 to 4000	check the master timer see note 4
U	TEMP OUT	- 099 to 099	check the temperature control % output.
u	% C IN	- 099 to 099	Check the Carbon control % output.
W	WAIT	0 to 471	wait for a flag
Y	IN C TEST	0 to 4000	check input C
Z	Z OPCODE	0 to 4000	unconditionally set loop 2 setpoint. Also used in conjunction with flag 11 to pass information to the background program. see notes.

NOTES:

1. The control loop on which the C and c OPCODES act is based on the instrument setup.
    a. If the control variable for Loop 1 is either CV0, CV1, or IN B, then these OPCODEs act on Loop 1.
    b. If Loop 1 is not CV0, CV1, or IN B but Loop 2 control variable is either CV0, CV1, or IN B, then these OPCODEs act on Loop 2.
    c. If neither loop is set for CV0, CV1, or IN B, these OPCODEs will act on Loop 1 (default).

2. The control loop on which the H and h OPCODEs act is based on the instrument setup (See Setup and Configuration: Parameters).
    a. If the control variable for Loop 1 is set for either IN A or IN C, these OPCODEs will act on Loop 1.
    b. If Loop 1 is not set for IN A or IN C, and Loop 2 is set for IN C, these OPCODEs will act on Loop 2.
    c. If Loop 1 is not set for IN A or IN C and Loop 2 is not set for IN C, then these OPCODEs will act on the Slave #1 Temperature controller.

3. The process factor that is affected is determined by the instrument setup (See Setup and Configuration: control variables) .
    a. If the controlled variable is CV0 (%Carbon) then PF 1 will be changed.
    b. If the controlled variable is CV1 (Dewpoint), then PF 2 will be changed.

4. The Programmer counts time in hours AND minutes. Time is entered and displayed as HH.MM where HH=hours and MM=minutes. The programmer uses timer F (TmrF) for counting the time.

5. The set flag opcode, **F**, works exactly like an event output opcode. There are twelve flags, numbered 0 to 11, which can be set or cleared as indicators to the background program. One flag, #11, also effects the operation of the **Z** opcode described later. The flags are set or cleared immediately and the limit opcode has no effect. The flags are physically stored in parameter **O0** in bits 4 through 15 for flags 0 to 11 respectively. These are the unused bits of the internal events setpoint.

The wait flag opcode, **W**, works exactly like an event input opcode. There are twelve flags, numbered 0 to 11, on which the wait can be specified for the flag to be set or cleared. If the selected flag is in the specified state the program continues. If the flag in not in the specified state the programs waits for the flag to be changed by the background program. A limit statement can be used specify the maximum wait time before a limit alarm will occur. The flags are physically stored in parameter **o0** in bits 4 through 15 for flags 0 to 11 respectively. These are the unused bits of the internal events actual values.

The set loop 2 opcode, **Z**, is used to specify the setpoint for loop 2. A special case arises if flag #11 is set by the **F** opcode. If flag #11 is set, then the **Z** opcode will set the Z1 accumulator instead of the loop 2 setpoint. This feature was added so that the Z1 could be set to values larger than 255. When flag #11 is set, the limit statement will have no effect following the **Z** opcode.

Otherwise, the limit statement will cause the program to wait for the loop 2 controlled variable to come within +/- 5 of the setpoint. Use the following sequence for these commands:

```
F   11.1      (turn on flag 11)
Z   600   (set ref num to 600)
```

If we want to ensure the background got it, add:

```
W    11.1 (wait for B Prog to acknowledge)
L    00.05 (but wait no more than 5 min)
F    11.0   (clear flag 11)
```

Special flags are also available to perform internal functions in the Dualpro when commanded to do so in a recipe. Setting flags 12, 13, 14, 15 are the same as setting internal events 0 through 4. Setting flag 16

“H” OPCODE REDIRECTION

The following OPCODES are effected by the redirection capability added to the version 3.5 programmer in the Multipro.
The 'H' OPCODE is used to set the setpoint of the temperature loop in a Version 3.5 recipe program. The 'h' OPCODE is used to perform an inquiry on the actual temperature of that same control loop.

a. The 'r' OPCODE ramps the temperature loop overtime.
b. The 'U' OPCODE is used to perform an inquiry on the temperature controllers percent output.

The following order of selection applied to the 'H' (and the 'h' 'r' & 'U') previously;

1. Loop 1 if it's controlled variable is Input A or Input C.
2. Loop 2 if it's controlled variable is Input C.
3. Slave temperature controller #1.

The redirection capability allows the operator to select whether the normal logic is used or the OPCODEs are directed to a specific instrument. The operator can then select to which channel and which temperature slave of that channel. Therefore, the 'H' OPCODE can be directed to a slave temperature controller even if loop 1 or 2 would have satisfied the above logic. Also, the 'H' OPCODE can be directed to the temperature slave of a slave Multipro.

Operator Selections

The operator has three selections to make in relation to the redirection of the 'H' OPCODE. These items are located in the '**Prog**' menu under the '**Setup**' key. The normal operator input uses of the keyboard applies to these selections.

1. The first selection is weather or not to activate the redirection. The upper display will show '**H OP**' and the lower display can be toggled between '**NORM**' and '**DIR**'. These

messages refer to normal or directed mode.

2. The second selection is the channel number where 0 refers to the master Multipro and numbers 1 through 15 its slave Multipros. The upper display will show '**H CH**' and the lower display a number from 0 to 15.
3. The final selection is the slave temperature controller of the selected channel. Since the slave Multipro may be a Protocol Convertor, the selection allows for sixteen temperature slaves. The upper display will show '**H SL**' and the lower display will show a slave from ' **H1** ' to ' **H16**'.

H Opcode Special Considerations

The limit statement can not be used after the 'H' OPCODE when it is in directed mode. If the limit statement is used it is ignored. The 'H' OPCODE usually require transmitting the setpoint over the network. If a limit statement is used in the normal fashion, the programmer would continuously execute the 'H' OPCODE quickly filling up the que. The limit can be used with the 'h' OPCODE since it does not transmit data. The ramp OPCODE, 'r', only sends the setpoint if it calculates a different value then was previously sent. However, it may be possible for a steep ramp to fill the communication que. This will cause a 201 alarm. The 201 alarm occurs whenever the communications que is full and the 'H' or 'r' OPCODEs attempt to send a setpoint.

The 'H' OPCODE does not wait for the actual setpoint to equal the set value for the same reason as above.

The 'H' and 'r' OPCODEs are effected by the write partition and the time-out partition the same as a version 4 OPCODE. Therefore, if the selected channel is in the write partition, the setpoint value will be written into the slave table.

The redirection feature does not know the difference between a Protocol Converter or a Multipro. The parameters will be read or written as defined for the sixteen slaves of a Protocol Converter. The Multipro only has eight temperature slaves and the other eight selections may be meaningless.

The redirection feature assumes all the selections are for communicating temperature controllers. If the selected temperature controller does not have a 2 in the low byte of the communications status parameter, an alarm 91 will occur.

Technical Information

The operator selections for the 'H' OPCODE redirection are located in parameter 8 of table 31. The information is bit mapped as follows:

Bit(s)	Description
0 - 3	Slave temperature controller; H1 = 0000 through H16 = 1111.
4 - 7	Channel number; 0 through 15
8	Mode; 0 = normal, 1 = directed
9 - 15	Not assigned

A version 4 program can access this information and change the assignments at any time.

The following table list the parameters associated with the sixteen slave temperature controllers.

Slave Channel	Comms Status	Setpoint	Actual Setpoint	Actual Process	Percent Output
H1	38H 56	40H 64	48H 72	50H 80	58H 88
H2	39H 57	41H 65	49H 73	51H 81	59H 89
H3	3AH 58	42H 66	4AH 74	52H 82	5AH 90
H4	3BH 59	43H 67	4BH 75	53H 83	BH 91
H5	3CH 60	44H 68	4CH 76	54H 84	5CH 92
H6	3DH 61	45H 69	4DH 77	55H 85	5DH 93
H7	3EH 62	46H 70	4EH 78	56H 86	5EH 94

Slave Channel	Comms Status	Setpoint	Actual Setpoint	Actual Process	Percent Output
H8	3FH 63	47H 71	4FH 79	57H 87	5FH 95
H9	C8H 200	D0H 208	D8H 216	E0H 224	E8H 232
H10	C9H 201	D1H 209	D9H 217	E1H 225	E9H 233
H11	CAH 202	D2H 210	DAH 218	E2H 226	EAH 234
H12	CBH 203	D3H 211	DBH 219	E3H 227	EBH 235
H13	CCH 204	D4H 212	DCH 220	E4H 228	ECH 236
H14	CDH 205	D5H 213	DDH 221	E5H 229	EDH 237
H15	CEH 206	D6H 214	DEH 222	E6H 230	EEH 238
H16	CFH 207	D7H 215	DFH 223	E7H 231	EFH 239

LIMIT STATEMENTS

There are various ways to force a program to wait for something to happen. Although it may seem that the specified condition should be easily satisfied, it is still wise to put realistic time limits on how long the wait should be.
The following table summarizes the effect the LIMIT statement has on each OPCODE.

OPCODE	LIMIT statement
C	LIMIT forces a wait and sets the maximum amount of time to wait for the process to reach the setpoint +/- 0.05 % Carbon (or setpoint +/- 5 for Dewpoint control).
E	Does not effect EVENT outputs. With EVENT inputs, it sets the maximum amount of time allowed for an EVENT to occur. The LIMIT statement here will cause the program to wait for the specified EVENT input to toggle to the specified state, either ON (.1) or OFF (.0), before proceeding. If this does not occur within the specified time, a LIMIT time-out alarm will occur. The EVENT input must be held in the specified state for at least 1 second to insure that the Programmer will acknowledge it. Thus, a momentary push button could not be used as an EVENT input unless some type of latching scheme is employed. If a LIMIT statement is not used, the program may wait indefinitely.
H	Forces a WAIT and sets the maximum time allowed for the temperature to reach SETPOINT +/-10°F (+/-5°C). The LIMIT statement here will cause the program to wait for the measured temperature to come within +/-10°F of the specified setpoint. If this does not occur within the specified time, a time-out alarm will occur. If no LIMIT statement is used, the Programmer simply sets the specified setpoint and goes on to the next STEP.
h	Sets the maximum time allowed for a condition to be met.
P	Forces a wait and sets the maximum amount of time to

OPCODE	LIMIT statement
	wait for the process to reach setpoint +/-.10 carbon.
r	ILLEGAL! The r OPCODE must always be followed by an H OPCODE.
S	Allows the soak timer to run only if the carbon is within a specified amount of the setpoint. LIMIT statement data is interpreted as %C (IE: L 0.05 means +/-.05%C deviation allowed).
W	The LIMIT statement here will cause the program to wait for the specified flag to toggle to the specified state, either ON (.1) or OFF (.0), before proceeding. If this does not occur within the specified time, a LIMIT time-out alarm will occur. If a LIMIT statement is not used, the program may wait indefinitely.

When a LIMIT statement follows any OPCODE not described above it is interpreted as a NOP.

The maximum time that a LIMIT statement will accept is 40 hours. Use the Programmer Status Display during a wait operation to reveal the time remaining before the timeout alarm will occur.

PROGRAMMER ALARMS

Programmer alarms can occur from several sources. Programmed alarms, 1 - 79, are used to indicate when some action is required by the operator. Deviation alarms, 81, 83, and 85, indicate potential problems with the process or sensors. The limit alarm, 93, also indicates a process problem. Alarms 91 and 92 indicate communications problems with the temperature buss or the events buss respectively. Other alarms indicate problems with the programs.

When a programmer alarm occurs, the ALARM DISPLAY page overrides the display. Either a message or the alarm number will be displayed on the screen. The messages either indicate the action required or the nature of the problem. Also, at the bottom of the screen the acronym ‘PAL’ (Programmer ALarm) is displayed followed by the alarm number.

Press any key or the knob and the ALARM ACKNOWLEDGE page is displayed. By rotating the knob, the operator can choose between ‘NO’, ‘YES, W/O SKIP’, or ‘YES, WITH SKIP’ The operator can either acknowledge the alarm with either ‘YES’ choice or not with the ‘NO’ choice. Then press ‘Enter’ or the knob to return to the normal display. If the alarm is not acknowledged, ‘PAL’ followed by the alarm number will continue to flash at the bottom of the

screen and any alarm contact configured for the programmer alarm will remain in the alarm state. In most cases either 'YES' choice will have the same effect. The alarm will be cleared, the 'PAL' at the bottom of the screen will disappear, and any alarm contact configured for the programmer alarm will return to the normal state. The 'SKIP' applies to a limit alarm, 93. If the choice is 'W/O SKIP', then the limit timer is reset and the program waits for the process the achieve the desired value. If the choice is 'WITH SKIP', then the program continues at the step following the limit statement.

The ALARM DISPLAY page only pops up automatically when an alarm occurs when there is a no alarm condition. If the previous alarm is not acknowledged, a new alarm will overwrite the old alarm and its number will be displayed at the bottom of the screen following the 'PAL'.

RECIPE PROGRAM ALARMS

ALARM	DESCRIPTION
81	"TEMPERATURE DEVIATION ALARM"
83	"%CARBON DEVIATION ALARM"
85	"THERMOCOUPLE VERIFICATION ERROR"
91	"TEMPERATURE CONTROLLER COMMUNICATIONS ERROR"
92	"OPTO BUSS COMMUNICATIONS ERROR"
93	"RECIPE LIMIT TIME OUT"
95	"RECIPE ERROR ILLEGAL RAMP"
96	"POWER FAILURE"
97	"RECIPE ERROR ILLEGAL SUBROUTINE CALL
98	"RECIPE ERROR BAD JUMP

LOGIC PROGRAM ALARMS

ALARM	DESCRIPTION
100	"Foreground; Bad load executing jump"
101	"Background; Bad load executing jump"
104	"Foreground; Step too big in jump"
105	"Background; Step too big in jump"
106	"Foreground; Prog number too big in jump"
107	"Background; Prog number too big in jump"
110	"Foreground; Bad load executing GOSUB"
111	"Background; Bad load executing GOSUB"
112	"Foreground; GOSUB too deep"
113	"Background; GOSUB too deep"
114	"Foreground; Step too big in GOSUB"
115	"Background; Step too big in GOSUB"
116	"Foreground; Program number too big in GOSUB"
117	"Background; Program number too big in GOSUB"

LOGIC PROGRAM ALARMS
ALARM DESCRIPTION (cont'd)

118	"Foreground; Bad load on return from GOSUB"
119	"Background; Bad load on return from GOSUB"
120	"Foreground; Bad load executing run program opcode"
121	"Background; Bad load executing run program opcode"
124	"Foreground; Step too big executing run program opcode"
125	"Background; Step too big executing run program opcode"
126	"Foreground; Prog number too big executing run program opcode"
127	"Background; Prog number too big executing run program opcode"
130	"Foreground; Read indirect too big"
131	"Background; Read indirect too big"
132	"Foreground; Write indirect too big or request queue full"
133	"Background; Write indirect too big or request queue full"
134	"Foreground; SYNC out but not master unit"
135	"Background; SYNC out but not master unit"
136	"Foreground; Send program to slave but not master unit"
137	"Background; Send program to slave but not master unit"
138	"Foreground; Start program in slave but not master unit"
139	"Background; Start program in slave but not master unit"
140	"Foreground; Program send value not valid"
141	"Background; Program send value not valid"
142	"Foreground; Data stack pushed too deep"
143	"Background; Data stack pushed too deep"
144	"Foreground; Data stack empty on pop"
145	"Background; Data stack empty on pop"
146	"Foreground; Block specification incorrect"
147	"Background; Block specification incorrect"
148	"Foreground; Block send but not a Master or que is full"
149	"Background; Block send but not a Master or que is full"
252	"Foreground; Bad load when restoring a program on a Power Fail Restore"
253	"Background; Bad load when restoring a program on a Power Fail Restore"
255	"Background; Bad load RPOG 1 on AUTO Prog Start"

SAMPLE RECIPE PROGRAMS

The following are some sample programs. These are provided to show how the Step/ OPCODE process works. These are examples only and not intended for any specific application. Please check your engineering to find the application programs best suited for you.

Program # 100

This program can be used to test the Dewpoint in a % output deviation.

Step	OPCODE	Mnemonic	Data	Description
1	u	Couti	90	Is output higher than 90% ? Indicating full gas.
2	B	bran	09.03	if yes, branch to step 9; if no branch to step 3.
3	u	Couti	-90	Is output higher than -90%? Indicating full air.
4	B	bran	05.12	If yes, branch to step 5; if no, branch to step 12.
5	c	Carbi	00.20	Is dewpoint higher than 20 %?
6	B	bran	07.15	If yes, branch to step 7; if no branch to step 15.
7	c	Carbi	00.50	Is dewpoint higher than 50%?
8	B	bran	15.01	If yes branch to step 15; if no branch to step 1.
9	E	EVENT	0.1	Turn on generator alarm.
10	A	ALARM	20	System is using too much gas.
11	B	bran	17.17	Branch to step 17.
12	E	Event	0.1	Turn on generator alarm.
13	A	Alarm	21	System is using too much air.
14	B	bran	17.17	Branch to step 17.
15	E	Event	0.1	Turn on generator alarm.
16	A	Alarm	22	Dewpoint deviation outside of 40 $\pm$ 10
17	E	Event	0.0	Turn off generator alarm.

Step	OPCODE	Mnemonic	Data	Description
18	I	Delay	120	Delay interval of 2 minutes.
19	B	bran	1.1	Branch to step 1.

This program forms a loop. It will be necessary to stop it to start another.

Program 200

This sample program used to be called the pre-load Carbon and temperature routine. It requires previously loaded programs to load the batch to be treated, sound alarms of a specified number code and operators to know what an "Alarm 79" or an "Alarm 78" is. The operators are expected to know how to respond to an alarm message.

See OPERATIONS MANUAL for specifics.

Step	OPCODE	Mnemonic	Value	Description
1	h	TEMPI	1400	Is temperature above 1400°F?
2	B	bran	03.05	If yes branch to step 3; If no branch to step 5.
3	h	TEMPI	1750	Is temperature above 1750°F?
4	B	BRAN	05.06	If yes branch to step 5; if no branch to step 6.
5	J	JUMP	174	sound alarm 79 and Quit.
6	c	CARBI	00.30	Is carbon above .3% ?
7	B	BRAN	08.10	If yes branch to step 8; if no branch to step 10.
8	c	CARBI	01.50	Is carbon above 1.5 % ?
9	B	BRAN	10.11	IF yes branch to step 10; if no branch to step 11.
10	J	JUMP	173	Sound alarm 78 and quit.
11	J	JUMP	149	Jump to loading routine.

Sample Program 3

The following program is a sample boost diffuse carburizing cycle. NO events or communicating temperature controller is involved. the program does not make use of LIMIT statements to guard against furnace malfunction in steps 2,4 or 6. The next program will cover those items.

Step	OPCODE	Mnemonic	Data	Description
1	C	CARBS	0.00	exclude enriching gas during start up.
2	h	TEMPI	1700	Is temperature above 1700°F?
3	B	BRAN	04.02	If yes branch to step 4; if no branch to step 2.
4	S	SOAK	2.00	Soak for 2 hours.
5	C	CARBS	1.20	Carbon setpoint of 1.2 %.
6	S	SOAK	9.00	soak for 9 hours.
7	C	CARBS	.80	Carbon setpoint .80 %.
8	S	SOAK	4.00	soak for 4 hours.
9	A	ALARM	01	cycle complete sound alarm to alert operator.

The soak times were for heating the load, boosting the carbon levels, diffusing the carbon through the atmosphere. The two carbon settings were to boost the carbon level and stabilize it.

Sample Program 4

The following program will demonstrate the protective functions of the communicating temperature controller and the LIMIT statement. Notice there are more opportunities for alarms to sound in this program. This should allow better control of the treating situation.

Step	OPCODE	Mnemonic	Data	Description
1	C	CARBS	0.00	Exclude enriching gas during heat up.
2	H	TEMPS	1750	Temperature setpoint of 1750°F.
3	L	LIMIT	3.00	wait 3 hours for furnace to reach ± 10°F, if not limit time out alarm 93 occurs.
4	S	SOAK	3.00	Soak for 3 hours while load reaches furnace temperature.
5	C	CARBS	1.20	Carbon setpoint 1.2%.
6	L	LIMIT	2.00	Wait 2 hours while Carbon reaches ± 10% C, if not limit time out alarm 93 occurs.
7	S	SOAK	8.00	Soak for 8 hours during boost carburizing.
8	C	CARBS	0.80	Reduce Carbon setpoint to .80% C. for diffusion.
9	L	LIMIT	0.50	Wait for the carbon to come within + 10 % of new carbon setpoint, if not time out alarm 93 occurs.
10	S	SOAK	3.00	Soak for 3 hours diffusion time.
11	A	ALARM	0012	Alarm sounds to notify operator that cycle has completed.

Sample Program 5

Sometimes more complex programming will be needed. The following program calls up programs as subroutines or jumps.

Step	OPCODE	Mnemonic	Data	Description
1	h	TEMPI	1400	Is temperature above 1400?
2	B	BRAN	03.05	If yes branch to step 3; if no, to step 5.
3	h	TEMPI	1750	is furnace temperature 1750°F?
4	B	BRAN	06.05	If yes, branch to step 6; if no branch step 5.
5	J	JUMP	174	Jump to program 174: sound alarm 79 and quit.
6	c	CARBI	00.30	Is Carbon above .3%?
7	B	BRAN	08.10	If yes branch to step 8; if no, to step 10.
8	c	CARBI	01.50	Is Carbon level above 1.5%?
9	B	BRAN	10.11	If yes branch to step 10; if no, to step 11.
10	J	JUMP	173	Jump to program 173: sound alarm 78 and quit.
11	G	GOSUB	149	Go to the loading subroutine and return.
12	h	TEMPI	1400	Is the temperature 1400°F?
13	B	BRAN	14.12	If yes branch to step 14; if no, to step 12.
14	S	SOAK	02.00	Soak the load for 2 hours while it comes up to heat.
15	r	RAMP	03.00	Ramp the temperature for 3 hours.
16	H	TEMPS	1750	Set the temperature to 1750 for the ramp.

DELETING A STEP:

(1) Go to the step to be deleted, using either [Enter] or [Shift].
(2) Press [Setup].

EXITING THE EDITOR WITHOUT SAVING THE PROGRAM:

(1) Press [Setpt] at any time to get to the SAVE display.
(2) Press [Setpt] again and the edited program is lost (the copy in non-volatile memory is unchanged).

EXITING THE EDITOR WITH PROGRAM SAVED:

(1) Edit through step 24 as required inserting NOP's wherever no operation is to be executed.

Or

(1a) Press [Setpt] at any time to get the SAVE display..
(2) Select the Program Number in the SET display using the Arrow keys until the desired Program Number is displayed.
(3) Press [Enter], the program is stored in that location previously is now lost.

NOTE: If SAVE display is showing and it is desired to change a step before saving, press the [Shift] key to back up to the desired step.

Displays and other Editing Notes

Since there is no room to display the step number while displaying the opcode, destination, and source; the program number being edited and the step number are flashed on the set and process displays before displaying the step data. Each press of the [Enter] or [Shift] keys which change the step number will cause the program number and step to be flashed.

The Process Display has Pnnn for program number.
The Set Display has S nn for step number.

Editing a Program

ENTERING THE PROGRAM EDITOR*:

(1) Press [Enter] + [Setpt] .
(2) Select the program to be edited using the Arrow keys.

(3) Press [Enter].
*Note that the unit will allow editing of a program that might be running. This editing will not affect the currently-running copy of this program until the program is actually restarted, either manually or by a program jump.

EDITING PROGRAM STEPS:

(1) Use the left and right arrow keys to select digits to be changed (either the opcode or the data).
(2) Use the up and down keys to change the selected digit.
(3) Press [Enter] to move forward to the next step.
Or
(3b) Press [Shift] to move backward to the previous step.

INSERTING A STEP:

(1) Go to the step where the new step is to be inserted, using either the [Enter] or [Shift] keys. The displayed opcode and data, as well as all higher step number opcodes and data, will be moved up one step when the step is inserted.
Note: information in step 24 will be lost.
(2) Press [Page Disp.].
(3) Enter the new step, opcode and data.
(4) Press [Enter].

When editing an opcode, if the left-hand mnemonic character is flashing then each press of the up or down arrow key will cause the opcode to change by 4 opcodes (see the table of editing order of opcodes). Likewise, if the right hand mnemonic character is flashing, then each press of the up or down arrow keys will cause the opcode to change by 1 unit. One unit could be the same opcode changing from a source parameter to source data (or vice versa) or changing to the next opcode. The difference between a source parameter or source data is shown in the SET display. If the source is a parameter, then the left two digits of the SET display are blank and the source code is shown in the right two digits. If the source is data, all four digits of the SET display will have characters in them.

The source data is normally in hexadecimal and each digit will continue to wrap around and affect only that digit. However, the source data can be edited in decimal by pressing the [Prog/Auto/Man] key, when one of the source digits is flashing (i.e. editing the source). In decimal mode however, each digit does not affect only itself, when the number wraps around, it affects the digit next to it because addition and subtraction are taking place. For example, if the right most digit were being edited and the up arrow was pressed past nine instead of the display being 0000, it would now be 0010.

A GOSUB to Program number zero, or a jump to Program number zero, loads and executes a blank (NOP) program.

PROGRAMMING ANALOG INPUTS

One of the features of the Multipro instrument is the ability to program a linear input such that it displays in the desired units. This is accomplished by using the input offset, input span, and input decimal point parameters. The input offset is the value to be subtracted from the input to achieve a zero display and has a range of -999 to 9999. The input span is a multiplying factor and has a range of -999 to 999. The range of the input span, when used in combination with the span's floating decimal feature, permits values whose accuracy ranges from +0.001 to +999. The input decimal point identifies where the decimal point should be placed on the display and has a value of 0 to 3, where 0 represents no decimal places (ie: +999) and 3 represents three decimal places (ie: +0.001). The following procedure should be followed to determine the proper values to enter into the input parameters.

1. Place the input selection in "Linear".
2. Select and display the “DATA” pages.
3. Set the voltage representing the zero value of the process either by adjusting the process to zero or by simulation. Adjusting the actual process is best.

   Note: the instrument will probably not be displaying zero.
4. On a sheet of paper, record the instrument reading as INZ.
5. Set the voltage representing the full scale value of the process in a similar manner as step at.
6. On the same sheet as above, record the instrument reading as INF.
7. Determine how the process units are to be displayed and to what accuracy. ie: 0 to 50 lph (liters per hour) could be displayed as 0 to 50, 0 to 50.0 or 0 to 50.00. Remember that if the process does not have the accuracy to justify a high resolution (ie: 01 lph) it should not be used. Also remember that the instrument displays only has 4 digits and if the value is negative, only 3 digits.
8. From the display range determined in step 7., determine the full scale counts (FSC) (the display without decimal points) and the decimal point location (IDP). In the step 7 example the 0 to 50 range would have an FSC of 50 and an IDP of 0. The 0 to 50.0 range would have an FSC of 500 and an IDP of 1 and 0 to 50.00 range would have an FSC of 5000 and an IDP of 2.
9. Calculate the input span (ISP) as the full scale counts divided by the difference in instrument readings (IE: ISP = FSC/(INF-INZ)).
10. Input the data into the instrument as follows:
    a. Set the input offset equal to INZ
    b. Set the input span equal to ISP.
    c. Set the input decimal point equal to IDP.
11. Change the input selection to program "PROG".
12. Vary the process over its range to test the accuracy of the values. Minor adjustments may be needed to achieve the desired result.
13. Record all values used and save them for future reference.
14. The results achieved will depend on how much rounding of numbers had to be done or how far ISP is from unity (1). Should questions arise, please contact MSI Customer Service for assistance.

PROGRAMMING ANALOG OUTPUTS .

The Multipro instrument allows the output range of the analog outputs to be specified by the user. This can be done providing that neither of the control loops' percent outputs have been selected. This is achieved by Entering the proper analog output offset and range value. The analog output offset is the value the selected variable achieves when the analog output is at its zero value (IE: zero volts or 4 mA). The analog output range is the difference between the value of the selected variable when the analog output is at full scale and the value of the variable when the analog output is at its zero value. This is demonstrated with a few examples. In each of the examples it is presumed that the analog output has previously been calibrated for 0 to 5 volts.

EXAMPLE NUMBER 1. Desire 0 to 1000°F to give 0 to 5 volt analog output. In this case the analog output offset value is 0 and the range value is 1000.

EXAMPLE NUMBER 2. Desire 1000 to 1250 mV to give 0 to 5 volt analog output. In this case the analog output offset value is 1000 and the range value is 250.

EXAMPLE NUMBER 3. A recorder has a 1 to 5 volt input range which is to represent 0 to 2000°F. In this case, it is necessary to interpolate at what value the analog output 0 would occur. Since 0 to 2000 is represented by a 4 volt range, the slope is 500° per volt. Therefore, the analog output zero would occur at 500° which is what is used for the analog output offset. The analog output range is the full scale value, 2000, minus the offset value, -500, which is 2500.

As shown by the examples, the programmable analog output can be programmed to not only provide the desired range, but to also compensate for differences in ranges of a recorder and the instrument.

MULTIPRO DATABASE

Introduction

The purpose of this document is to define a database for the Multipro. This data base is to allow the background program a means of saving information in a structured format for later retrieval.

Specifications

Number of files:	4
Maximum size of file:	16k bytes (8196 parameters); files 0 and 1 share the same memory space and files 2 and 3 share the same space. The maximum space for both files 0 and 1 is 16k (likewise for 2 and 3). Therefore if file 0 is 16k, file 1 cannot be used.
Maximum number of records:	Limited by the 16k byte maximum file size and the number of fields per record.
Maximum number of fields:	240; limited by the size of a parameter table.
Size and type of field:	1 version 4 parameter (2 bytes).
Database operations:	Define file, check status, erase file, delete file, change number of records, write record, read record, search, read fields, erase record, read structure and write fields.

Version 4 programmer

All database operations are performed with one opcode, 'DM', Data Manager. The operations of this opcode are controlled by the source data, a control block, and sometimes the destination data. At the completion of the opcode, an error code is written into the destination parameter and the flags set. An error code of 0 (which means the zero flag is set) indicates the successful completion of the operation. The other error codes are defined later.

The source data defines the operation and the file number. The file number (0 to 3) is in the high byte and the operation code in the low byte. The operation codes are as follows:

Code	Description
0	Read file status; error code is 0 if file valid, 1 if not. If the destination value is non-zero, it defines the starting parameter of the control block.
1	Define file; Number of records and fields determined by the control block. The destination value defines the starting parameter of the control block. If the size of the file would over-write an existing file or exceed memory, the file is defined as large as possible and an error code is returned. The file is erased (data zeroed) when defined.
2	Erase file; This function zeros all data in a defined file without changing its structure.
3	Delete file; This function deletes the file by making it invalid.
4	File re-size; This function adjusts the size of the file by changing the number of records. The destination value defines the starting parameter of the control block. If the number of records is less than the old size then data in the records removed is lost. If the number of records is increased, the data in the added records is undefined. An error is returned if the size of the new file would over write an existing file or exceed memory space. The number of fields in the control block must agree with the existing value.
5	Write record; The record specified by the control block is written starting at the parameter specified.
6	Read record; The record specified by the control block is read into the parameter table starting at the parameter specified.

7 Search; The control block specifies the starting record, the field, the condition code, the search value, and the search mask. The search mask must be FFFFH for numeric searches. The condition codes are: 0 = zero, 1 = not zero, 2 = equal, 3 = greater than, 4 = less than, and 5 = not equal.

8 Read fields; This function reads one field from a series of records. The control block specifies the starting record number, the field number, the starting parameter in which to write, and the number of records to read.

9 Erase record; Sets all data in the specified record to zero.

10 Read structure; The number of records and number of fields is read into the control block. If the destination value is non-zero, it defines the starting parameter of the control block.

11 Write fields; This function writes one field to a series of records. The control block specifies the starting record number, the field number, the starting parameter from which to read the data, and the number of records to write.

The control block consists of seven parameters. Since the control block is only used when the 'DM' opcode is executed, the same control block could be used for all four files. It is also possible to use more than one control block per file by changing the control block specification with the status read function. Only the file define, status read, file re-size, and read structure functions change the control block starting location. The seven parameters in the control block are defined as follows:

Offset	Definition and use
0	Record number; Defines the file size for the define and change number of records functions. Defines the starting record for the search function and is set to the record that meets the criteria. Defines the starting record for the read fields function and is set to the record following the last read. Defines the record of operation for the read record, write record, and erase record functions.
1	Field number; This value sets the number of fields per record in the define file functions. This specifies the field of operation for the search, read fields, and write fields functions.
2	Condition code; This value is the condition code for the

search function. The condition code is ignored for all other functions. The condition codes are: 0 = zero, 1 = not zero, 2 = equal, 3 = greater than, 4 = less than, and 5 = not equal.

3 Value; This parameter specifies the value for comparison in the search function. This also specifies the number of records to read in the read fields function or to write in the write fields fucntion.

4 Mask; This parameter specifies a bit mask for the search operation allowing individual bits or bit combinations to be searched. Since the Data Manager does not distinguish between numeric and logical searches, the mask must be set to FFFFH (65535) for numeric searches.

5 Read parameter; This value is the first parameter into which the read function will store a record. The high byte is the table number and the low byte the parameter number. The read function will not cross a table boundary; therefore, the number of parameters in the table following the start point must be equal to or greater than the number of fields per record.

6 Write parameter; This value is the first parameter from which the write function will store into a record. The high byte is the table number and the low byte the parameter number. The write function will not cross a table boundary; therefore, the number of parameters in the table following the start point must be equal to or greater than the number of fields per record.

Error Codes

The error codes are stored in the destination register at the completion of the function. The branch or jump based on the zero flag can follow the 'DM' opcode to invoke an error handling routine. Two type of errors exist, fatal and non-fatal. A fatal error indicates that the function was aborted when the error was detected. A non-fatal error means that the function was completed but the outcome may have been modified. All fatal errors will have bit 7 set (i.e. be >= 128 (80H)). The following table lists the error codes and their causes.

ERROR CODES

Code	Description	Possible causes
0	Success	Function was executed.
1	File is invalid	This error only occurs with the read file status function and indicates that the file is undefined.
2	Insufficient memory	The size of file requested by the define file function or re-size file function was larger than the available memory. The file was successfully defined or re-sized but with fewer records. The number of records is written into the control block.
3	End of file	The end of the file was reached before the search condition was satisfied.
F0 (240)	Invalid file number	The file number specified in the high byte of the source data is greater than 3. Only 0, 1, 2, and 3 are valid file numbers.
F1 (241)	Invalid function	The function number specified in the low byte of the source data is not defined.
F2 (242)	Bad control block spec	The control block starting parameter is not allowed because it is not valid, too close to the end of the table, or in the reserved area of table 0. Possible causes; invalid table number (>31), parameter number greater than 233, Z register greater than Z9 (249), or table 0 parameter less than 120 (78 hex).
F3 (243)	File not valid	The file specified has not been defined. This error is returned by all functions, except file

		define and status read, when the file has not been defined.
F4 (244)	Invalid record number	For most functions this error is returned if the record number specified in the control block is equal to or greater than the number of records in the file. Since the record numbers are zero based, a file with 100 records contains numbers 0 to 99. For the file define or re-size functions, this error occurs if the number of records specified in the control block is 0 or exceeds 16384.
F5 (245)	Invalid parameter	This error occurs if the parameter specified in the control block is; too close to the end of the table for the number of fields, in a table number greater than 31, or less than parameter 120 (78 hex) in table 0.
F6 (246)	Invalid field number	For most functions this error occurs if the field number specified in the control block is equal to or greater than the number of fields per record for the file. Since the field numbers are zero based, if there are 10 fields per record they are numbered 0 through 9. For the file define function, this error occurs if the number of fields in the control block is 0 or greater than 240. For the re-size file function, this error indicates the number of fields in the control block is different than the file being re-sized.
F7 (247)	Not enough memory	This error only occurs if the memory available is insufficient to define a file with even 1 record. In other words, the memory available divided by the number of fields is less than 1.
F8 (248)	Invalid condition code	This error is returned by the search function when the condition code specified in the control block is undefined.

BASIC

BASIC will have access to the same four files as the version 4 programmer. BASIC

has several functions and statements to access the files. BASIC has a fixed control block and record buffer for each file. The record buffers are a fixed length of 240 parameters. BASIC can read and write all 240 parameters of each buffer at any time. However, only the quantity corresponding to the number fields per record are read from or written to a file.

The following BASIC statements apply to database operations.

Statement	Description
GET file,record	Read the specified record from file into its buffer.
PUT file,record	Write the file buffer to the specified record of the file.
FIND file,field,value,condition,mask,record	Search the file for the value in the specified field based on the condition code and mask, starting at the specified record. The condition codes are: = (equal), < (less than), > (greater than), and n (not equal). The mask is used for bit or byte manipulation and must be hex FFFF for numeric searches.
DBA file,records,fields	Creates a file of the number of records and fields per record as specified.
DBC file1,file2	Copies the record buffer of file2 into the record buffer of file1.
DSET file,field,data	Stores the data value into the specified field of the record buffer of the file.

The following BASIC functions apply to the database operations:

Function	Description
FST(file)	Returns the file status of the specified file, 0 for a valid file or 1 for a non-valid file.
DBF(file)	Returns the number of fields per record in the specified file.
DBR(file)	Returns the number of records in the specified file.
DBE(file)	Returns the last error that occurred on the specified file from a BASIC operation.

DBL(file) Returns the current record position in the specified file.

DPARM(file,field)
Returns the data value of the specified field in the record buffer of the file.

There are two levels of error checking on the file operations. BASIC will generate a syntax error if the file number is greater than 3, the field is greater than 239, missing parameters, or other typos. The syntax errors will stop the program unless an ONERROR statement was executed. The database handler also generates error codes. These do not stop the BASIC program. The code is stored in the control block for each file for retrieval with the DBE(file) function. The errors for each file do not effect other files. A successful operation writes a zero in the error location to indicate it was completed. The error codes are the same as described for the version 4 programmer.

There are no OPEN and CLOSE statements associated with the file operations. Any of the four files can be accessed with the GET or PUT statements at any time. Also there is no file or record locking. BASIC and the version 4 programmer or free to trash each others data at will. It is recommended that a field be designated just for hand-shaking between BASIC and the version 4 programmer. For example, a value of 0 would indicate no data in this record. A 1 would indicate that the version 4 programmer has placed raw data in this record for BASIC to process. A 2 would indicate that BASIC has processed the data and placed the results in the record. A 3 could then indicate that the version 4 programmer has retrieved the processed data.

The record buffers for files 0 through 3 are actually parameter tables 53 through 56 respectively. The version 4 programmer can not access tables above 31. However, the BASIC SETPAR statement and PARM() function can access all the tables, i.e. 0 through 56. Therefore, tables 32 through 52 are available for BASIC program usage.

Logic Language Programmer

Program Layout

A Logic Language program consists of twenty-four (24) steps. There are two hundred (200) user definable programs in the instrument. There can also be up to 50 fixed system programs.

An OPCODE (operation code) tells the program interpreter what to do with the destination and the source. Each step of a program must have an OPCODE. Most OPCODEs require a destination (IE: what is being acted upon) and a source (IE: the information needed). The program control OPCODEs are slightly different. The program control OPCODEs do not use a destination, but interpret it as a condition code. The OPCODE is displayed to the instrument operator as a two character mnemonic; however, internally it is stored as an eight (8) bit byte. One bit is reserved to specify whether the source is a parameter specification or a data word. Another bit specifies whether the destination is in table 9 or table 16. With six bits to specify OPCODEs it is possible to have one hundred twenty-eight (128) different OPCODEs (these are currently over 50 OPCODEs).The table provides a description of each OPCODE in the Logic Language programmer.

The next part of a program step is the destination. The destination is usually an instrument parameter specification; however, the program control OPCODEs (BRANCH, JUMP, GOSUB, etc.) use a condition code instead. The destination is displayed to the instrument operator as a two character mnemonic; however, internally it is stored as an eight (8) bit byte which allows two hundred fifty-six (256) destinations to be specified. Refer to the instrument parameter section for a detailed description of each instrument parameter. Most OPCODEs change the information in the destination parameter, where the result of the operation is being stored. The program control OPCODEs do not use a destination, but interpret it as a condition code. The low two bits of the destination byte is decoded into one of the four condition codes: unconditional (UN), less than (LT), greater than (GT), or zero/equal to (ZE).

The last part of the step is the source. The source may be either a parameter specification or a data word. When the source is a parameter specification it is of an identical format as the destination. When it is a data word, the source is a sixteen (16) bits 2's complement binary integer. This means that the source can have a value ranging from -32,768 to +32,767. The source parameter is never changed by any OPCODE except the exchange OPCODE and is only used as data or reference.
A program step requires an eight bit OPCODE, an eight bit destination, and a sixteen (16) bit source for a total of thirty-two (32) bits (ie: four bytes). A program requires twenty-four (24) steps of four bytes each plus four bytes of overhead for a total of one hundred (100) bytes. Therefore, the two hundred (200) user definable programs require twenty thousand (20,000) bytes of nonvolatile memory.

The following list will clarify the opcodes for Multipro and the Dualpro.

MULTIPRO EQUIVALENTS OF DUALPRO VER 4 OPCODES

Multipro	Dualpro
NO OPER	--
ADD	AD
EXT ANALOG	AI
AND	AN
ABSOLUTE	AV
BIT TEST	BI
BRANCH NOT	BN
BRANCH	BR
BANK SELECT	BS
COPY BLOCK	CB
CLOCK READ	CK
COMPLEMENT	CO
COMPARE	CP
DEC BR NOT	DB
DECREMENT	DE
DELAY	DL
DATA BASE	DM
DATA SAVE	DS
DIVIDE	DV
EXCHANGE	EX
GOSUB NOT	GN
GOSUB	GS
HOLD PROG	HP
NCREMENT	IN
INDEX READ	IR
INDEX WRITE	IW
JUMP NOT	JN
JUMP	JP
KILL BASIC	KB
KILL PROG	KI
MULT/DIVIDE	MD
MULTIPLY	ML
NEGATE	NG
OR	OR
PULL DEST	PD
POP STACK	PS
RUN BASIC	RB
READ INDIR	RD
RESET BIT	RE
ROTATE LEFT	RL

Multipro	**Dualpro**
RAMP	RP
ROTATE RGHT	RR
RUN SLAVE	RS
RUN PROG	RU
SET CLOCK	SC
SAVE DEST	SD
SET BIT	SE
SHIFT LEFT	SL
SEND PROG	SP
SHIFT RIGHT	SR
SEND SYNC	SS
STORE	ST
SUBTRACT	SU
SYNC	SY

```
WRITE INDIR WR
XOR         XR
```

Alphabetical Summary of Program OPCODEs

OPCODE	Name	Flag	Description
AD	Add	Y	DEST = DEST + SRC
AI	Analog Input	Y	Read external analog raw data
AN	And	Y	Logically AND DEST and SCR, result in DEST
AV	Absolute Value	Y	DEST = | DEST |
BI	Bit Test	Y	Test bit in DEST, bit number in SRC
BN	Branch Not	T	Branch to a step based on condition not true
BR	Branch	T	Branch to a step based on condition true
BS	Bank Select	N	Select bank of Z resisters
CB	Copy Block	Y	Copy Block
CK	Clock	Y	Reads clock parameters
CO	Complement	Y	Logically complement DEST
CP	Compare	Y	Compare DEST to SRC
DB	Dec Bran	N	DEC DEST and branch to SRC step if result not 0
DE	Decrement	Y	Decrement Destination
DL	Delay	Y	Pauses programs for short period of time
DM	DataBase	Y	DataBase Manager
OPCODE	Name	Flag	Description
DS	Data Save	Y	Data Save
DV	Divide	Y	DEST = DEST/SRC
EX	Exchange	T	Exchanges source and DEST data
GS	Gosub	T	Call a subroutine based on condition true

GN	Gosub Not	T	Call a subroutine based on condition not true
HP	Hold Program	Y	Holds program based on condition
IN	Increment	N	Increments destination
IR	Indexed Read	N	Stores into DEST value of parameter at SRC plus value of FX (IF FX=4, IR(SP), (Z0)=ST(SP), (Z4))
IW	Indexed Write	T	Stores SRC into parameter at DEST plus value of FX (If FX=1, IW(SP), (Z0)=ST(SX),(Z0))
JP	Jump	T	Jump to program based on condition true
JN	Jump Not	N	Jump to a program based on condition not true
KB	Kill Basic	N	Kill (stop) basic program
OPCODE	Name	Flag	Description
KI	Kill	Y	Kill (stop) background program
ML	Multiply	Y	DEST = DEST * SRC
MD	Multiply/Divide	Y	DEST = DEST * FX/SRC
NG	Negate	N	DEST = -DEST
_	NOP	Y	No operation
OR	Or	N	Logically OR DEST and SRC, result in DEST
PD	Pull DEST	N	Pull DEST from data stack
PS	Pop Stack	N	Pop last call out of program stack
RB	Run Basic	N	Run basic program
RD	Read	N	Read Indirect, DEST = (SRC)
RE	Reset Bit	Y	Reset bit in DEST, bit number

			in SRC
RL	Rotate Left	N	Rotate DEST left bit out left into right
RP	Ramp	Y	Ramp DEST to SRC value over time in ACCO
RR	Rotate Right	N	Rotate DEST right bit out right and left
RS	Run Slave	N	Run program in slave, SRC = Number and Step
RU	Run	N	Run program, SOURCE = Number and Step
OPCODE	Name	Flag	Description
SD	Save DEST	N	Save (push) DEST on data stack
SE	Set Bit	N	Set bit in DEST, bit number in SRC
SC	Set Slave Clocks	Y	Sends the master's time to the slaves' clocks
SL	Shift Left	N	Shift DEST left by SRC time, insert zeros
SP	Send Program	Y	Send program, SRC=Number to send and Number to save
SR	Shift Right	N	Shift DEST right by SRC times, insert zeros
SS	Send Sync	N	Send sync commands to slaves
ST	Store	Y	Store SRC to DEST
SU	SUB	N	DEST = DEST - SRC
SY	SYNC	N	Wait for sync flag
WR	Write	Y	Write indirect, (DEST) = SRC
XR	XOR		Logically XOR DEST and

SRC, result in DEST

Note:

1. DEST = DESTINATION REGISTER
2. SRC = SOURCE REGISTER OR DATA VALUE
3. Y = FLAGS CHANGED, N = NO EFFECT, T = TESTED

Command Definition and Format

The following table lists each command, the definition of the command and the format.

OPCODE Description

AV ABSOLUTE VALUE

Definition: The ABSOLUTE VALUE replaces the destination parameter with its absolute value. This OPCODE does not require a source.

Format: AVdd dd is the destination code

AD ADD

Definition: The ADD OPCODE adds the source to the destination using sixteen bit signed integer arithmetic.

Format: ADdd ADdd dd is the destination code

xxxx or SS XXXX is the data

ss is the source code

AI ANALOG INPUT

Definition: Specified by the source, and places the data in the destination; changing the flags.

Format: AIdd AIdd dd is the destination

xxxx or SS XXXX is the data

SS is the source code

AN AND

Definition: The AND OPCODE performs a bit by bit logical AND on the destination and source and places the result in the destination.

Format: ANdd ANdd XXXX is the data code xxxx or SS SS is the source code dd is destination

Command Definition and Format (Continued)

OPCODE Description

BI BIT TEST

Definition: The BIT TEST OPCODE examines the destination bit, specified by the source, and if a logical one sets the greater than flag (GT) and if a logical zero sets the zero/equal flag (ZE), the destination is unchanged.

Format: BIdd dd is the destination code.

xxxx XXXX is the data

SS is the source code

Example: The BIT TEST is used in conjunction with the BRANCH and BRANCH NOT functions as follows:

BI dd XXXX

BR ZE 'for BIT = 0

or BN ZE 'for BIT = 1

BN BRANCH NOT

Definition: The BRANCH NOT OPCODE is the same as the BRANCH except that the condition code must NOT be satisfied to execute the BRANCH. There is one exception, a BRANCH NOT UNconditional will always execute just like the BRANCH UNconditional.

Format: BRdd BRdd dd is the modifier code, XXXX or SS which may be one of the following:

GT (greater than)
LT (less than)
ZE (zero/equal)
UN (unconditional)
XXXX is the data
SS is the source code

Command Definition and Format (Continued)

OPCODE Description

BR BRANCH

Definition: The BRANCH OPCODE is used to transfer program control within a program.

Format: BRdd BRdd dd is the modifier code, XXXX or SS which may be one of the following:

GT (greater than)
LT (less than)
ZE (zero/equal)
UN (unconditional)
XXXX is the data
SS is the source code
BS (blank select)

BS Bank Save

Definition: The BANK SAVE OPCODE is used to change Z register banks. There are 16 banks (0 - 15) of register Z8 thru ZF.

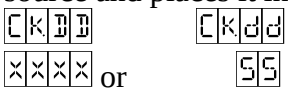
Format: BRdd BRdd The condition code must be XXXX or SS satisfied For the bank select to occur.

Command Definition and Format (Continued)

OPCODE Description

CK CLOCK

Definition: The CLOCK OPCODE takes the real time clock parameter specified by the source and places it in the destination; changing the flags.

Format:  dd is the destination code

XXXX is the data

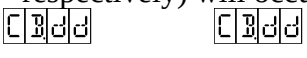
SS is the source code

Source Value	Parameter
0	Year 0-99, represents 1980-2079
1	Month 1-12
2	Day 1-31
3	Hour 0-23, 0=midnight, 23= 11:00 pm
4	Minute 0-59
5	Second 0-59
6	Day of Week 1-7, 1=Sunday, 7=Saturday
7	Not Assigned

CB Copy Block

Definition: The COPY BLOCK OPCODE will copy the block specified by the source into the block specified by the destination. The block is specified by the block number in the low byte and the channel number (slave table) in the high byte. Only block numbers 0 through 9 are valid except for channel 0 where 16 through 25 are also valid. The extended parameters (bank 2) are specified by block numbers 16 through 25. Valid channel numbers 0 through 15 where 0 is the instruments own table. An invalid block specification, either source or destination, will cause an error 146 or 147 (foreground or background respectively).

The copy block can also send a block to a slave instrument. The destination block is specified as above and then the slave channel address is placed in the high nibble (i.e. channel address *4096). If the instrument is not a master or the communication que is full, then an error 148 or 149 (foreground or background respectively) will occur.

Format:  XXXX is the data code  or  SS is the source code dd is destination

Command Definition and Format (Continued)

OPCODE Description

CO COMPLEMENT

Definition: The COMPLEMENT OPCODE performs a bit by bit complement of the destination (IE: a 0 becomes a 1 and a 1 becomes a 0). No source is required.

Format: COdd dd is the destination code

CP COMPARE

Definition: The COMPARE OPCODE changes the condition code flags depending on the relationship of the destination to the source. Neither the destination nor the source are changed by this OPCODE. If the destination is greater than the source, then the GT flag is set. Likewise, if the destination is less than the source, then the LT flag is set; if the source and destination are equal, then the ZE flag is set.

Format: CPdd CPdd dd is the destination code

XXXX or SS XXXX is the data SS is the source code

DB DECREMENT BRANCH NOT ZERO

Definition: The DECREMENT BRANCH NOT ZERO OPCODE subtracts one from the destination. If the result is not zero then program control is passed to the step specified by the source. This is a very powerful OPCODE for loop counting.

Format: JJdd JJdd dd is the destination code

XXXX or SS XXXX is the data

SS is the source code

Command Definition and Format (Continued)

OPCODE Description

DE DECREMENT

Definition: The DECREMENT OPCODE subtracts one from the destination parameter value. No source is required.

Format: 

dd is the destination code

DL DELAY

Definition: The DELAY OPCODE is used to pause the program for a short period of time. The source is the delay time in seconds whose maximum is 250. Any value greater than 250 is treated as if it were 250. If the condition code is not satisfied, then no delay occurs.

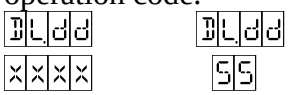
Format: 

dd is the modifier code, which may be one of the following:

GT (greater than)
LT (less than)
ZE (zero/equal)
UN (unconditional)
DS (data save)

DM DATA MANAGER

Definition: The DATA MANAGER opcode calls the database program. The destination parameter would receive the error code or if no error the source data is the operation code.

Format: 

dd is the modifier code,

Command Definition and Format (Continued)

OPCODE Description

DS DATA SAVE

Definition: The DATA SAVE OPCODE will push the source data onto the data stack (different from program stack).

Format:

dd is the destination code

DV DIVIDE

Definition: The DIVIDE OPCODE divides the destination by the source using sixteen bit signed arithmetic.

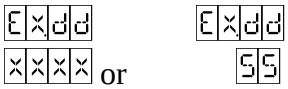
Format:

dd is the destination code
XXXX is the data
SS is the source code.

EX EXCHANGE

Definition: Exchanges the source for the destination (ie: St Z1 10, St Z2 13, EX Z1 Z2, Z1=13, Z2=10). Source and destination values won't matter, they simply swap. No effect on flags. If the source is data then the data is stored in the destination, all the destination is host.

Format:



dd is the destination
code XXXX is the data
SS is the source code

Command Definition and Format (Continued)

OPCODE Description

GS GOSUB

Definition: The GOSUB works in the same manner as the JUMP except that the present program number and step are saved on the program stack before the jump is executed. When the new program ends, it will RETURN to the original program at the step following the GOSUB and continue from there.

Format: GSdd GSdd dd is the modifier code,
XXXX or SS which may be one of the following:
GT (greater than)
LT (less than)
ZE (zero/equal)
UN (unconditional)
XXXX is the data
SS is the source code

GN GOSUB NOT

Definition: The GOSUB NOT OPCODE is the same as the GOSUB except that the condition code must NOT be satisfied to execute the GOSUB. There is one exception, a GOSUB NOT UNconditional will always execute just like to GOSUB UNconditional.

Format: GNdd GNdd dd is the modifier code, XXXX or SS which may be one of the following:
GT (greater than)
LT (less than)
ZE (zero/equal)
UN (unconditional)
XXXX is the data
SS is the source code

Command Definition and Format (Continued)

OPCODE Description

HP HOLD PROGRAM

Definition: The HOLD PROGRAM OPCODE, 'HP', is used to place a program in hold or release a program from hold. When executed from a foreground program, the background program is placed or released from hold. Likewise, when executed from a background program, the foreground program is placed or released from hold. The HP OPCODE is conditional and operates only if the condition is true. The source data is used to determine if a program is to be placed in hold or released from hold. If bit 0 of the source data is 1 then the other program is placed into hold and if 0 the other program is released from hold. The other bits of the source data have no effect. The hold condition is identical to placing a program in hold from the front panel. The operator can release a program from hold regardless of how it was placed in hold. Likewise, a program can release another program from hold regardless of how it was placed in hold. If a program attempts to place a program in hold that is already in hold there is no effect, IE: it stays in hold. If a program attempts to release a program from hold which is already in run mode, there is no effect. In other words, no damage is done by multiple attempts:

Format: HPdd HP.dd dd is the modifier xxxx or SS code, which may be one of the following:

LT (less than)
ZE (zero/equal)
UN (unconditional)

Command Definition and Format (Continued)

OPCODE Description

IN INCREMENT

Definition: The INCREMENT OPCODE adds one to the destination parameter value. No source s required.

Format: INdd dd is the destination code --|--|--|--

IR INDEXED READ

Definition: The INDEXED READ OPCODE is similar to the STORE OPCODE in that it is used to store the value of the source into the destination. However with the INDEXED READ, the value stored in the (FX) parameter will be added to the address of the source parameter prior to the store operation. (i.e. If (FX)=2 and a IR (Z0), (R0) is performed, it would be equivalent to ST (Z0), (R2).) The high byte of FX is the table number and the low byte is a parameter offset.

Format: IRdd IRdd XXXX is data --|--|--|--

or SS SS is source data

IW INDEXED WRITE

Definition: The INDEXED WRITE OPCODE is similar to the STORE OPCODE in that it is used to store the value of the source into the destination. However with the INDEXED WRITE, the value stored in the (FX) parameter will be added to the address of the destination parameter prior to the store operation. (IE: If (FX)=2 and a IW (Z0), (R0) is performed, it would be equivalent to ST (Z2), (R0).) The high byte of FX is the table and the low byte is a parameter offset.

Format: IWdd IRdd dd is destination --|--|--|-- or

SS xxxx is data ss is source

code

Command Definition and Format (Continued)

OPCODE Description

JP JUMP

Definition: The JUMP OPCODE is used to transfer program control to another program. The source is the program number (either data or the contents of a parameter) which will be loaded and run. The low byte is the program number and the high byte, if not zero, is the step number. If the high byte is zero then the program is started at step one. The JUMP OPCODE is executed only if the condition code is satisfied, otherwise, it is treated a NOP.

Format: JPdd JPdd dd is the modifier XXXX SS code, which may be one of the following:
GT (greater than)
LT (less than)
ZE (zero/equal)
UN (unconditional)
XXXX is the data
SS is the source code

JN JUMP NOT

Definition: The JUMP NOT OPCODE is the same as the JUMP except that the condition code must NOT be satisfied to execute the JUMP. There is one exception, a JUMP NOT UNconditional will always execute just like the JUMP UNconditional.

Format: JNdd JNdd dd is the modifier code, XXXX or SS which may be one of the following:
GT (greater than)
LT (less than)
ZE (zero/equal)
UN (unconditional)
XXXX is the data
SS is the source code

Command Definition and Format (Continued)
OPCODE Description

KB KILL BASIC

Definition: The KILL BASIC OPCODE forces a control-C into the BASIC interpreter input buffer. If the condition code is not satisfied, it is treated as a NOP.

dd is the modifier code, which may be one of the following:
GT (greater than)
LT (less than)
ZE (zero/equal)
UN (unconditional)

KI KILL

Definition: The KILL OPCODE will stop a program if the condition code is satisfied. If the KILL OPCODE is executed in a foreground program, the background program is stopped. Likewise, if the KILL OPCODE is executed in a background program, the foreground program is stopped.

Format:

dd is the modifier code, which may be one of the following:
GT (greater than)
LT (less than)
ZE (zero/equal)
UN (unconditional)
XXXX is the data
SS is the source code

ML MULTIPLY

Definition: The MULTIPLY OPCODE multiplies the source times the destination using sixteen bit signed arithmetic and stores the result in the destination.

Format:


or


dd is the destination code
XXXX is the data
SS is the source code

MD MULTIPLY/DIVIDE

Definition: The MULTIPLY/DIVIDE multiplies the destination times the factor (parameter FX) producing a 32 bit signed result. The 32 bit result is then divided by the source producing a 16 bit signed result which is then stored in the destination.

Format:


or


dd is the destination code
XXXX is the data
SS is the source code

Command Definition and Format (Continued)
OPCODE Description

NG NEGATE

Definition: The NEGATE OPCODE replaced the destination parameter with its 2's complement (IE: the negative value). This OPCODE does not require a source.

Format:  dd is the destination code

NOP NO OPERTATION

Definition: The NOP is no operation code. It's only purpose is to take up space in a program. Since the programs are of fixed length (twenty-four (24) steps), the unused locations must have something in them and that is the purpose of a NOP.

Format:  N/A

OR LOGICAL OR

Definition: The OR OPCODE performs a bit by bit logical OR on the destination and source and places the result in the destination.

Format:   dd is the destination code

 or  XXXX is the data

SS is the source code

PD PULL DESTINATION

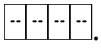
Definition: The PULL DESTINATION OPCODE pulls a value off of the data stack and stores it into the destination parameter. (See the SAVE DESTINATION, SD). There is a separate stack for foreground and background programs.

Format:  dd is the destination code 

PS POP STACK

Definition: This is an advanced program feature which removes the calling program return information from the program stack, therefore, the program will not return to the calling program.

Format:  N/A

 .

Command Definition and Format (Continued)

OPCODE Description

RB RUN BASIC

Definition: The RUN BASIC OPCODE forces a RUN command into the BASIC interpreter input buffer if a BASIC program is not running. The SRC value is the line number at which the BASIC program will start. If bit 15 of the source is set, then BASIC will run without clearing its variables. If bit 14 the source is set, then BASIC will continue from where it was stopped.

Format: RBdd xxxx or RBdd ss

dd is the modifier code, which may be one of the following:
GT (greater than)
LT (less than)
ZE (zero/equal)
UN (unconditional)
xxxx is the data
ss is the source data.

RD READ INDIRECT

Definition: The READ INDIRECT OPCODE is like a STORE OPCODE except that the source value is interpreted as an address and the parameter specified by that address is then stored into the destination. For example, if the source was a data word of value six (6), the value of the sixth parameter will be stored in the destination not the value six (6). If the high byte of the source is not zero, it is interpreted as a channel number (IE: channel number times 256). This allows the network master to access parameters from the slave instruments.

Format: RDdd xxxx or RDdd ss

dd is the destination code
XXXX is the data
SS is the source code

OPCODE DESCRIPTION

RE RESET BIT

Definition: The RESET BIT OPCODE forces a bit in the destination to a logical zero. The source specifies which bit, 1-15.

Format: REdd xxxx or REdd ss

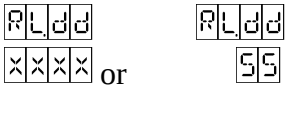
dd is the destination code
XXXX is the data
SS is the source code

Command Definition and Format (Continued)

OPCODE Description

RL ROTATE LEFT

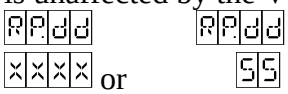
Definition: The ROTATE LEFT OPCODE rotates the destination sixteen bit value to the left. The source specifies how many bit positions to shift. The high order bits shifted out of the word are inserted in the low order bits.

Format: 

dd is the destination code
XXXX is the data
SS is the source code

RP RAMP

Definition: The purpose of a RAMP is to change a parameter at a constant rate over a period of time. The ramp generator needs to know which parameter to ramp, what the ending value should be and how long to take. Since this is three pieces of information and only two items can be specified in a program step, the RAMP requires that the time, in minutes, be stored into accumulator 0 (Z0) before the RAMP step. The destination specifies which parameter will be ramped and the source is the ending value. Once a ramp has been started accumulator 0 (Z0) and the source, if a parameter, can be changed. The RAMP OPCODE sets up and starts the ramp generator. This program does not stop at the RAMP and wait, therefore, the program is free to perform other operations. Only one RAMP can be operating at a time, therefore, if another RAMP OPCODE is executed before the ramp generator has finished, the first ramp is aborted and the new one started. If the program should end, or should the instrument be removed from program mode, the ramp will be aborted. If the foreground program is in Ver 4 mode, only a foreground program can initiate a ramp. If the foreground program is in Ver 3.5 mode, the background program can initiate a ramp. This ramp does not effect and is unaffected by the Ver 3.5 ramp.

Format: 

dd is the destination code
XXXX is the data
SS is the source code

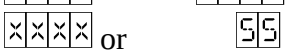
Command Definition and Format (Continued)

OPCODE Description

RR ROTATE RIGHT

Definition: The ROTATE RIGHT OPCODE rotates the destination sixteen bit value to the right. The source specifies how many bit positions to shift. The low order bits shifted out of the word are inserted in the high order bits.

Format:



or

dd is the destination code
XXXX is the data
SS is the source code

RS RUN SLAVE

Definition: The RUN SLAVE OPCODE will start a program running in a slave instrument. The value in the destination is the slave channel number, the source low byte is the program number, and the source high byte is the starting step number (zero will start at step one). The program will run in the foreground of the slave unless bit 15 of the source is set will start a background program.

Format:

or

XXXX is the data
SS is the source code

Command Definition and Format (Continued)

OPCODE Description

RU RUN

Definition: The RUN OPCODE will start a program at the program number and step specified in the source (same format as the JUMP). If the condition code is not satisfied, it is treated as a NOP. A foreground program will start a background program and background program will start a foreground program.

Format: RUdd RUdd XXXX or SS dd is the modifier code, which may be one of the following:

GT (greater than)
LT (less than)
ZE (zero/equal)
UN (unconditional)
XXXX is the data
SS is the source code

SD SAVE DESTINATION

Definition: The SAVE DESTINATION OPCODE will push the value of the destination parameter onto the data stack (different from the program stack), this allows the destination parameter to be used for some other purpose. (See the PULL DESTINATION, PD.) There is a separate stack for the background and foreground programs.

Format: SDdd dd is the destination code ----

Command Definition and Format (Continued)

OPCODE Description

SE SET BIT

Definition: The SET BIT OPCODE forces a bit in the destination to a logical one. The source specifies the bit number. The bit number must be from zero to fifteen where zero is the least significant bit.

Format: SEdd SEdd dd is the destination code
XXXX or SS XXXX is the data
SS is the source code

SC SET SLAVE CLOCKS

Definition: The SET SLAVE CLOCKS OPCODE commands the communications processor to set the Real Time Clocks of the slave instruments to the time of the sending instrument. If the sending instrument is not a master, or if the condition code is not satisfied, then the OPCODE is treated like a NOP.

Format: SCdd ---- dd is the modifier code, which may be one of the following:
GT (greater than)
LT (less than)
ZE (zero/equal)
UN (unconditional)

SL SHIFT LEFT

Definition: The SHIFT LEFT OPCODE shifts the destination sixteen bit value to the left. The source specifies how may bit positions to shift. The high order bits shifted out of the word are lost. Binary zero's are inserted in the low order bits.

Format: SLdd SLdd dd is the destination code
XXXX or SS XXXX is the data
SS is the source code

SP SEND PROGRAM

Definition: The SEND PROGRAM OPCODE will select a program and send it to a slave instrument. The destination must contain the slave channel number, the source low byte is the program number to send, and the source high byte is the program number to save it under in the slave. If bit 15 of the destination is set then a version 3.5 program will be sent, otherwise a version 4 program is sent.

Format: SPdd SPdd dd is the destination code
XXXX or SS XXXX is the data
SS is the source code

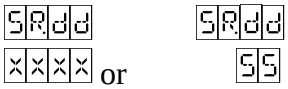
Command Definition and Format (Continued)

OPCODE Description

SR SHIFT RIGHT

Definition: The SHIFT RIGHT OPCODE shifts the destination sixteen bit value to the right. The source specifies how many bit positions to shift. The low order bits shifted out of the word are lost. Binary zero's are inserted in the high order bits.

Format:

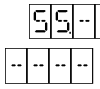


dd is the destination code XXXX is the data SS is the source code

SS SEND SYNC

Definition: The SEND SYNC OPCODEs requests that the communications handler send a SYNC command to the slaves. The program waits until the communications handler acknowledges sending the SYNC command. An error occurs if the instrument running a program with a SEND SYNC OPCODE is not a network master.

Format:

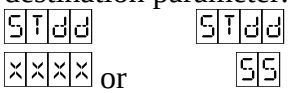


N/A

ST STORE

Definition: The STORE OPCODE takes the source value and places it in the destination parameter.

Format:

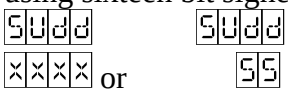


dd is the destination code XXXX is the data SS is the source code

SU SUBTRACT

Definition: The SUBTRACT OPCODE subtracts the source from the destination using sixteen bit signed arithmetic.

Format:



dd is the destination code XXXX is the data SS is the source code

Command Definition and Format (Continued)

OPCODE Description

SY SYNC

Definition: The SYNC OPCODE causes the program to wait until a SYNC command is received from the network master. This allows for multiple instruments to have their programs synchronized.

Format: SS-- N/A

WR WRITE INDIRECT

Definition: The WRITE INDIRECT OPCODE is like a STORE OPCODE except the destination register value is interpreted as an address and the source value is stored into the parameter specified by that address. If the high byte of the destination value is not zero, it is interpreted as a channel number. This allows the network master to change parameters in the slave instruments.

Format: WRdd dd is the destination
XXXX code XXXX is the
data SS is the source code

XOR EXCLUSIVE OR

Definition: The XOR OPCODE performs a bit by bit logical XOR on the destination and source and places the result in the destination.

Format: RIdd RIdd dd is the destination
XXXX or SS code XXXX is the
data SS is the source code

BASIC Interpreter

AN INTRODUCTION TO BASIC

With the proper connections to a terminal, the Multipro can provide a BASIC interpreter with the access to instrument parameters, ability to display messages on the instruments, and ability to perform more complicated operations not available within the instrument's programmer. This interpreter allows for full length programs to be written and run, or for single commands to be typed in and executed. As soon as the instrument is turned on and the start-up statement appears on the terminal screen, BASIC is ready and available. (If the BASIC port is set to BASIC. See Setup and Configuration for details.)

Those who have some knowledge of BASIC may choose to determine which commands are available for this version of BASIC, and then proceed. For those who wish to review, or for those who are unfamiliar with BASIC, the first few sections provide a brief introduction.

Program Lines

Programs consist of "lines" of statements and comments. The structure of these lines can be very strict in some programming languages, and more flexible in others. In BASIC, the structure is generally as follows: nnn BASIC statement.

a. "nnn" indicates the line number. The line number is used as a reference point for branching and editing. If it is necessary, an interval of ten or more is placed between lines to allow for the addition of other lines later. IE: the first line is usually numbered 10, and the second line 20. This allows for the addition of a line at 15, or any other integer number. This is necessary if it was discovered that a line was forgotten or if an additional line was needed. Line numbers can range from 1 to 9999.

b. BASIC statements are the BASIC commands and related information used to make up the program. Statements may be either executable or non-executable. Executable statements tell BASIC what to do and when. Non-executable statements do not cause any program action. Examples of non-executable statements are REM. and DATA. (REM. is a remark statement which allows for comments to be added to the program without interfering with the actual running of the program.) Both the executable and the non-executable commands are explained in detail later.

BASIC Characters and Reserved Words

BASIC uses both the lower and upper case letters, the digits 0 through 9, and the list of characters in the following Table :

BASIC Characters

Character	Name/Description
=	equal sign or assignment symbol
+	plus sign or concatenation symbol
-	minus sign
*	asterisk or multiplication symbol
/	slash or division symbol
\	backslash

^	caret or exponentiation symbol
(	left parenthesis
)	right parenthesis
%	percent sign
#	number sign
$	dollar sign or string type declaration
!	exclamation point
&	ampersand
,	comma
.	period or decimal point
'	single quotation mark (apostrophe)
;	semicolon
:	colon or statement separator
?	question mark (PRINT abbreviation)
<	less than
>	greater than
"	double quotation mark or string delimiter
_	underline

Certain words have special meaning to BASIC and are therefore "reserved". Reserved words include all BASIC commands, statements, function names, and operator names.

Constants and Numbers

Constants are the values that BASIC uses during the execution of the program. Constants can be string (or character) constants, or they can be numeric constants.

String constants are sequences of up to 255 characters enclosed by double quotation marks, and can be similar to any of the following forms:

a. "HELLO" or "Hello"
b. "You may have WON $1,000,000!!"
c. "Please stand by"

Numeric constants can be positive or negative numbers, cannot contain commas, and can be entered in one of the following forms:

a. Floating point: Both positive and negative numbers are represented in exponential form (similar to scientific notation). The constant consists of an optionally signed number (negative numbers need a - sign, positive numbers do not need a + sign), with or without a decimal point, followed by the letter E and an optionally signed exponent. The E is typically read as "times ten to the power of", and the exponent must not contain a decimal point. The following examples should help to clarify the form of the floating point constant:

Floating Point Constants

Number	Floating Point Constant
256	2.56E2 or 25.6E1 or 256E0
.003456	3.456E-3
-1,000,000	-1E6
-.0567	-5.67E-2

Note

Floating point constants range from 1.2E-38 to 6.8E+38 (can also be written as 6.8E38).

b. Hex: Numeric constants can be entered as hexadecimal numbers with up to four digits, and preceded by an x. The hexadecimal digits are the numbers 0 through 9, A, B, C, D, E, and F.

Hexadecimal Constants

Number	Hexadecimal Constant
256	x100
50	x32
672	x2A0
10,000	x2710

c. Octal: Numeric constants can be entered as octal numbers with up to six digits, preceded by a zero. Octal numbers only use the digits 0 through 7.

Octal Constants

Number	Octal Constant
256	o400
50	o62
672	o1240
10,000	o23,420

Note

All numbers in this version of BASIC are floating point, single precision (have seven or fewer digits), and are in exponential form.

Variables

Variables are names used to represent values within a BASIC program. There are two types of variables: numeric and string. The numeric and string variables work similar to the numeric and string constants. Numeric variables may only contain a value that is a number, and a string variable may only contain a character value.

The length of a string variable is not fixed, and can be anywhere from 0 to 255 characters, as determined by the string value assigned to the string. String variables are initially assigned as having zero characters, and therefore a zero length.

Numeric and string variables can be set to a constant, read in from data input statements, or be the results of calculations within the program. The variable type must match the type of data being assigned to it. Numeric variables are initially 0.

Naming Variables

BASIC variable names can be composed of a single letter, or a single letter and a single number. The first character of the variable name must be a letter. Special characters used to identify the type of variable are allowed as the last character (IE: $ used to indicate a character string, as in L.$).

Declaring Variable Types

A variable's name determines its type. All variable names are considered to be numeric by

default unless the last character of the name is a $, indicating a string character. The dollar sign is a variable type declaration character. It "declares" that the variable will represent a string.

Arrays are variables that contain a group, or table of values under one name. Each individual value in the array is called an element. Arrays must have the name and number of elements in the array specified in a statement. These are known as the defining, or dimensioning statement (DIM).

Each array element is named with the array name subscripted with a number or with numbers. An array variable name has as many subscripts as there are dimensions in the array. The subscript indicates the position of the element in the array (zero is the lowest position).

The following statement creates a one dimensional array that contains 30 elements: 10 DIM L.(29)

This version of BASIC provides for one dimensional arrays, and for arrays dimensioned with two variables in numeric constants only; string variables cannot be arrayed. Arrays cannot have more than 4095 elements; therefore, the maximum one dimensional array is 4095 (IE: DIM L.(4094)). Also if the program is large and/or a large number of variables are defined, an out of memory error may occur.

Numeric Expressions and Operators

Numeric expressions can be simple constants or variables. They can also be combined by a numeric operator to produce a numeric value. There are four categories of numeric operators: arithmetic, functions, logical, and relational.

Arithmetic Operators

Arithmetic operators are the operators which perform the basic arithmetic functions. The operators are listed in the table below with examples (order of operations will be explained later).

Arithmetic Operators

Operation (BASIC)	Operator	Example	Example
^	Exponentiation	$2^3=8$	2^3
-	Negation	-5	-5
*	Multiplication	2 x 3=6	2*3
/	Division	6÷3=2	6/3
+	Addition	2+3=5	2+3
-	Subtraction	5-3=2	5-3

Numeric functions act like "short cuts" because they are designed to perform arithmetic functions with little work. IE: if the square root of 121 was desired, considerable work and programming space would be needed. However, with the use of the BASIC function SQR, the square root can be obtained with just one program line: 10 A=SQR(121). Each function acts as a command and is therefore listed and explained with the commands. The result of a function

must either be assigned to a variable (A=SQR(121)), or printed (PRINT SQR(121)).

Logical Operators

Logical operators are based on Boolean algebra and perform logic operations on numerical values. Logical operators work on 16 bit binary integer numbers from 0 to 65535. Numbers greater than 65535 are treated as if they were 65535, and numbers less than 1 are treated as if they were 0. The logical operators are &(AND), @(OR), and %(XOR) and are most easily explained by the following examples:

a. &(AND): The & operator performs a bit by bit logical ANDing of the numbers. "Bit by bit" indicates that a binary form of the number is necessary in order to properly determine the outcome. If two bits in the same position are both 1's, then the result will be a 1. If either bit is 0, then the result is 0.

Note: All three tables have the same numerical value, but only the binary demonstrates the actual process which determines the outcome of the & operator.

&(AND) Examples

Binary:	0010 0111 0001 0000
	& 0000 0001 0010 1100
	= 0000 0001 0000 0000
Decimal:	10000 & 300 = 256
Hex:	x2710 & x12C = x100

b. @(OR): The @(OR) operator performs a bit by bit logical ORing of the numbers. If two bits in the same position are both 0's, then the result will be a 0. If, however, one or both bits are 1, then the result will be a 1.

@(OR) Examples

Binary:	0010 0111 0001 0000
	@ 0000 0001 0010 1100
	= 0010 0111 0011 1100
Decimal:	10000 @ 300 = 10044
Hex:	x2710 @ x12C = x273C

c. %(XOR): The % operator performs a bit by bit logical XORing of the numbers. If two bits in the same position have the same value (i.e. both are 1's or both are 0's), then the result is 0. If they have two different values, then the result is 1.

%(XOR) Examples

Binary:	0010 0111 0001 0000
	% 0000 0001 0010 1100
	= 0010 0110 0011 1100
Decimal:	10000 % 300 = 9788
Hex:	x2710 % x12C = x263C

Relational Operators

Relational operators are operators that compare two values. String values as well as numeric values can be compared with relational operators providing that all values being compared are of the same type. That is, a string must be compared with a string, and a numeric with a numeric. The relational operators are as follows:

Relational Operators

Operator	Relation Tested
=	Equality, also used to assign variable values
<> or ><	Inequality
<	Less than
>	Greater than
<= or =<	Less than or equal to
>= or =>	Greater than or equal to

Numeric comparisons can compare numeric values such as X and Y, or they can be used to compare numeric expressions, such as X+Y and X*Y/3. String comparisons compare string values in an alphabetic fashion. Lowercase letters are considered greater than uppercase letters, and all letters are considered greater than numbers that are contained in the string (numbers not contained in a string cannot be compared with a string). If one string has a smaller number of characters than another, then it is considered to be less than the other string. Blank spaces do affect the comparisons: "UP " is greater than "UP", because of the extra space. Unless a variable is used to represent them, all string constants must be enclosed in quotation marks when being compared. IE: if X$="hello" it can be compared in any of the following ways with another string:

a. X$="hello"
b. X$ > "hi"
c. "hello" > "hi"
d. "hello" > Y$

Order of Execution

When performing numeric operations, it is important to understand the order in which BASIC executes each operation. For example, one might think that 2+3*4 would equal 5*4 or 20 when in actuality, the operation equals 2+12 or 14. The order in which BASIC executes operations, or the order of operations, is as follows:

a. Operations contained within parentheses ( )
b. Function calls are evaluated first (IE: the SQR command)
c. Arithmetic operations are performed next, in the following order:
   I. ^
   ii - (negation)
   iii. *,/ (in the order they appear from left to right)
   iv. +,- (in the order they appear from left to right)
d. Relation operators
e. Logical operators, in the following order:
   I. & (AND)

ii. @ (OR)
iii. % (XOR)

BASIC Speed Hints

Two programs that perform the same operation but are written in different ways can perform at different speeds. This is because of the manner in which BASIC carries out commands. For programs to perform as quickly as possible

a. Define any constants used in the program as variables. ASCII to floating point is a slow process.
b. Define the most used numeric variables first. Variables are stored and searched in the order defined except for string variables which move as their length changes.
c. Place the most commonly used subroutines near the beginning of the program. BASIC starts at the beginning to search for line numbers.

BASIC Commands

The first part of this section contains an alphabetical summary of the BASIC commands. The second part will contain each command listed separately, explained in more detail, and with examples.

Summary of Commands, Functions, Statements, & Operators
(In Alphabetical Order)

ABS(x)	Absolute value of x
& (Logical AND)	Logical AND as bit mask determination. for <expr>&<expr>
AND	Used in the IF...THEN statements to logically AND conditions
ATN(x)	Arctangent of x (x in radians)
AWAKE	Awake an instrument from Sleep Mode
BFCLR	Clears the communications input buffer
BGET(x)	Gets a value from protected memory
BLOCK	Locks the BASIC program so that it will not be inadvertently changed.
BON	Break ON, allows CNTLC to stop program execution
BOFF	Break OFF, prevents CNTL-C from stopping program execution
BSAVE	Stores a value in protected memory
BUNLOCK	Unlocks the BASIC program so that it can be changed.
CLEAR	Clears all variables
CLOSE	Closes the communications port
CLS	Clears the screen (sends an ANSI escape sequence to the terminal)
CNTL-C	Stop execution of current program
COS (x)	Cosine of x (x in radians)
COL x,y,z	Set screen colors and attributes
CONT	Continues execution after STOP or break
CPH	Continues execution after clearing Program Hold Flag
CUP x	Cursor up x lines
CDW x	Cursor down x lines
CRT x	Cursor right x char.
CLF x	Cursor left x char.

Summary of Commands, Functions, Statements, & Operators
(In Alphabetical Order)
(Continued)

CLN	Clear to end of line
DATA value, value,	Provide sequential data for READ command
DBA file, records, fields	creates a file of the number of records and fields per record as specified.
DBC file 1, file 2,	Copies the record buffer of file 2 into the record buffer of file 1.
DBE (0)	Returns the last error that occurred on the specified file from a BASIC operation.
DBF (0)	Returns the number of fields per record in the specified file
DBL (0)	Returns the current record position in the specified file
DBR (0)	Returns the number of records in the specified file.
DIM var(x, y)	Provide dimensional range and allocate for numeric arrays
DPARM (file, field)	Returns the data value of the specified field in the record buffer of the file.
DSET	

file, field, data Stores the data value into the specified field of the record buffer of the file.

ECHK (0)	Checks the status of a message sent out the Events Port
EDIT 'nn'	Allows the editing of line 'nn'
END	Signifies the end of program execution
EOF(0)	Returns a 0 if a complete message has been received thru the communications port, otherwise returns a 1
EREC(0)	Gets a reply from the Events Port
ERR$(0)	Returns a string which contains the line number and the error
ESEND	Sends a message out of the Events Port
EXB	Executes an instrument background program
EXF	Executes an instrument foreground program
EXP(x)	Calculates the value of e to power x
FOR var= (STEP <expr>)	FOR loop and counter with <expr> to expr> optional step
FIND file, field, value, condition, mask, record	Search the file for the value in the specified field based on the condition code and mask, starting at the specified record. The condition codes are = (equal),< (less than), > (greater than), and n (not equal). The mask issued for bit or byte manipulation and must be hex FFFF for numeric searches.
FPAR (number)	Returns a pair of instrument parameters assuming that they are in floating point format.
FRE (0)	Returns the amount of available memory
FREE	Prints the amount of available memory
FSET number, value	Sets a pair of instrument parameters in floating point format.
FST (file)	Returns the file status of the specified file, 0 for a valid file or 1 for a non-valid.
GET file, record	Read the specified record from a file into its buffer
GOSUB <line #>	Call routine at <line #>
GOTO <line #>	Transfer control to <line #>

Summary of Commands, Functions, Statements, & Operators
(In Alphabetical Order)
(Continued)

HEX$(n)	Returns a hex string of the number n value from 0 to 65535 (to a value from 0 to FFFF)
IF <expr>(THEN) ...(ELSE)	Remainder of program line up to ELSE is executed if statement(s) <expr> is true, otherwise the program line after ELSE is executed
INCHK(0)	Returns a status byte of operator input operation
INFLOAT	Initiates operator input mode with floating point
INGET(0)	Returns the current value of the operator input
ININT	Initiates operator input mode with integers
INKEY$(0)	Returns a one character string of the last key pressed at the terminal or a null string if no key was pressed
INLOGIC	Initiates operator input with yes or no response
INPUT var	Read value of variable from console
INQUIT	Cancels the operator input mode
INT(x)	Integer value of x
(LET) var=<expr>	Assign value of <expr> to variable
LIST	LIST current program to the terminal
LIST (line a, line b)	LIST current program to terminal from line a to line b
LOC(0)	Returns the number of characters received thru the communications port
LOG(x)	Calculates logarithm to base 10 of (x)
LN(x)	Calculates natural logarithm x
LPRINT	Print command for a “line printer” device
MGET( )	Receives a data transfer command to either store the values received from the BASIC Port (instrument) or sends instrument parameter values to the instrument
MPROC( )	Processes the data received from MGET
NEW	Initializes BASIC; clears variables, pointers
NEW*	Initializes BASIC; retains variable values
NEXT var	Ending expression of FOR loop counter
ON x GOTO L., 12, I3,...	Transfer control to line lx based on the value of x
ON x GOSUB l1, I2, I3,...	Call routine at line lx based on the value of x

Summary of Commands, Functions, Statements, & Operators
(In Alphabetical Order)
(Continued)

ONERROR lll	Defines the line number to send program control, should an error occur

OPEN X,Y	Opens the communications port for BASIC to use. Refer to the communications section for details
@ (LOGICAL OR)	Logical OR for bit <expr>@<expr>determination and setting
OR	Used in IF...THEN statements to logically OR conditions
PM0$(A$)	Returns a string for the instrument page display
PM1$(A$)	Returns a string for the instrument page display
PM2$(A$)	In operator input mode, updates a message
PARM(num)	Returns the instrument parameter specified
PDATA (number, step)	Returns the data value of recipe program, n, step, s.
PEDIT	Invokes the instrument program editor
PEEK (addr)	Returns the decimal value at address (addr)
POKE addr, value	Stores value at address (addr)
POPCODE (n, s)	Returns a one character sting which is the OPCODE of a recipe program number, n, step, s.
POS l,c	Position the cursor at line l and column c
POS (0)	Returns the current line print position.
PRINT <expr> or ? <expr>	Prints the value of <expr> to the terminal
PRINT or ?	Sends CHR$(13) and CHR$(10) to terminal
PUT file record	Write the file buffer to the specified record of the file.
RDUMP	Displays a block of instrument parameters
READ var, var,...	Assign data to variables from DATA statement
REC	Reads a string from the communications buffer if the message is complete
REM.	Remark statement (no operation)
RESTORE	Resets DATA statement pointer to initial value
RETURN	Return from subroutine
RND(X)	Generates random number between 0 and X.
RUN	Executes the current program
SDLRC	Send with Delimiter and LRC; transmits a string out the communications port after appending a delimiter, LRC, and EOT.
SEND	Transmits a string out the communications port
SETPAR num, value	Sets an instrument parameter to the specified value
SGN(x)	Sign of x (-1 or +1)
SIN(x)	Sine of x (x in radians)

Summary of Commands, Functions, Statements, & Operators
(In Alphabetical Order)
(Continued)

SIO	Causes the terminal UART to be reset up
SLEEP	Puts BASIC port into sleep mode
SLSTAT (0)	Returns the sleep (1) or awake (0) status of an instrument

SPB x	Sets a program break point
SQR (x)	Square root of x
SSP	Single steps a foreground program
STOP	Terminates program execution.Returns to COMMAND mode. CONT will restart at point where execution was terminated
TAN (x)	Tangent of x (x in radians)
XEQ	Execute the current program without zeroing variables
% (Logical XOR) (expr>%<expr>	Logical XOR function
+,-,*,^	Arithmetic operators for add, subtract, multiply, divide and exponentiation in floating point arithmetic
>,<,=,>=,<=,<>	Comparison operators (used with IF)

String Functions

ASC(str)	Returns the decimal value of the first ASCII character in the specified string
DAY$(0)	Returns the calendar day MM-DD-YYYY
CHR$(x)	Returns a single ASCII character by converting the decimal value of x to ASCII
GET$(n)	Gets n characters from the Communications Port
INPUT A$	Inputs a text string; quotation marks removed; no spaces allowed
INPUT LINE A$	Same as INPUT A$ but embedded spaces and quotation marks allowed
INSTR(x, A$,B$)	Searches for B$ in A$ beginning at position x; returns the beginning position of B$ in A$ (or 0 if not found)
LEFT$(A$,n)	Returns a string that consists of the leftmost n characters of A$
LEN(A$)	Returns the number of characters in A$
MID$(A$,C, n)	Returns a string of n characters from A$ beginning at character position C
NUM$(x)	Converts the number x to a string with a leading space or - sign
RIGHT$(A$,n)	Returns a string that consists of the rightmost n characters of A$
SPACE$ (n)	Returns a string of n ASCII spaces
STRING$(n, d)	Return a string of n ASCII characters whose decimal value is d
TIME$(0)	Returns the time of day (HH:MM:SS.S)
VAL(A$)	Convert numeric string to numeric value
+	Concatenate (join) strings
>,<,=,>=, <=, <>	Comparison operators (used with If)

Explanation of Commands, Functions, Statements & Operators
(In Alphabetical Order)

ABS
(Function)

Definition: A mathematical function that returns the absolute value of a numeric expression. The absolute value is always positive or zero.

Format: Y=ABS(x) x may be any numeric expression.

Example: Y=ABS(x)

For x =	Y will =
3	3
-3	3
0	0
-500	500
500	500

$(AND)
(Logical Operator)

Definition: A logical operator used for bit by bit comparison. If two bits in the same position are both 1's, then the result will be a 1. If one or both bits are not 1, then the result is 0.

Format: <expression> & <expression>
The expressions are numeric.

Example: Binary: 0010 0111 0001 0000
& 0000 0001 0010 1100
= 0000 0001 0000 0000

Decimal: 10000 & 300 = 256

Hex: x2710 & x12C = x100

All three of the above examples have the same numerical value, but only the binary demonstrates the actual process which determines the outcome of the & operator.

Notes:
The key word AND is used in IF...THEN statements to logically AND conditions. See the IF...THEN statement for more details.

ASC
(Function)

Definition: Returns the decimal value of the first ASCII character in the specified string.

Format: Y=ASC(str)
str is the specified string of which the first ASCII character is taken.

Example: Y=ASC(X$)

For X$ =	Y=
Test	84
test	116

ATN
(Function)
Definition: A mathematical function that returns the arctangent of a numeric expression. The arctangent is the angle whose tangent will produce the given numerical expression.
Format: Y=ATN(x) x may be any numeric expression. Y will be given in radians between (-PI/2) and (PI/2). Radians may be converted to degrees by multiplying by (180/PI). p is equal to 3.14159.
Example: Y=ATN(x)

For x =	Y= (in radians)	Y= (in degrees if converted)
-10^50 (or any large #)	-PI/2	-90
10^50 (or any large #)	PI/2	90
.5774	PI/6	30

(In Alphabetical Order)
(Continued)

AWAKE
(Statement)

Definition: Used in either a program, or as a single command to awaken an instrument for use with the BASIC interpreter. When used in conjunction with the SLEEP statement, multi-dropped instrument communication becomes more convenient.

Format: AWAKE n :n is the address of the instrument to be awakened. The AWAKE command is used with the @ symbol to access instruments that are in sleep mode.

Example: If the instrument at address 3 is in sleep, then the following command is used to wake it: @AWAKE 3

BFCLR
(Statement)

Definition: Clears the communications buffer in order that no extraneous data is left to become part of a new message.

Format: BFCLR

Example: 110 BFCLR
120 SEND A$

BGET
(Function)

Definition: Gets a numeric value from protected memory. Protected memory consists of 4608 values of memory that are normally not accessible by any process, and are not cleared in BASIC.

Format: A=BGET(n)
Where n=0 to 4607 is the location

Example: 20 Y = BGET(1)

BLOCK
(Statement)

Definition: BASIC PROGRAM LOCK keeps the BASIC program from being accidentally changed.

Format: BLOCK

Example: BLOCK OR 10 BLOCK

Explanation of Commands, Functions, Statements & Operators
(In Alphabetical Order)
(Continued)

BOFF
(Statement)

Definition: Break OFF, inhibits a CNTL-C from stopping execution of a program.
Format: BOFF
Example: BOFF or 10 BOFF
Notes:
The break off option is available so that binary data can be received on the BASIC port without program disruption.
Since there is not a way to stop the program from the BASIC port when the break is off, this should be used with caution. The only way to stop the program is to run an instrument program with the KB OPCODE.

BON
(Statement)

Definition: Break ON allows a CNTL-C to stop program execution.
Format: BON
Example: BON or 10 BON

BSAVE
(Statement)

Definition: Save a numeric value into protected memory.
Format: BSAVE n, m
Where: n=0 to 4607 is the location number and m is the value to be stored.
Example: 30 BSAVE 3,COS(x)

BUNLOCK
(Statement)

Definition: Unlocks the BASIC program so that it can be changed.
Format: BUNLOCK
Example: BUNLOCK OR 10 BUNLOCK

Explanation of Commands, Functions, Statements & Operators
(In Alphabetical Order)
(Continued)

CHR$
(Function)

Definition: Converts a number to its ASCII character equivalent.
Format: Y$=CHR$ (n): n is a numeric value in the range 0 to 255.
Example: Y$=CHR$ (n)

for n=	Y$=
84	T
116	t

CLEAR
(Statement)

Definition: Clears all variables.
Format: CLEAR
Example:
```
10 DIM A(20): ' Define array with 21 elements
.
.
.
100 CLEAR: "Clear variables"
110 DIM A(30): "A" is redefined with 31 elements
```

CLOSE
(Statement)

Definition: Closes the communications port and releases it for use by the instrument.
Format: CLOSE
Example: 10 CLOSE

CLS
(Statement)

Definition: Clears the screen.
Format: CLS
The CLS statement also sends the cursor to the home, or upper left hand corner, position. The terminal being used must recognize the ANSI escape sequence (Vt-100 emulation) in order for this statement to work.
Example: If the screen has been filled with excess text, or if the current screen is confusing, a CLS will produce a blank screen and start at the upper left hand corner.

CONT
(Command)

Definition: Resumes program execution after a break has been made.
Format: CONT
The CONT command is usually used to resume program execution after CTRL-C has been pressed, or a STOP statement has been executed. Execution will continue at the point the break occurred.
Example: A program is in the middle of execution and CTRL-C is pressed:
Stop at Line 40
Ok
CONT
and the program continues execution at line 40.

COS
(Function)

Definition: A math function that returns the cosine of an angle given in radians.
Format: Y=COS(x) : x is the angle expressed in radians. Degrees may be converted to radians by multiplying the degrees by p/180. (p is equal to 3.14159)
Example: Y=COS(X)

CPH
(Statement)

Definition: Clears the Program Hold Flag and allows a program to continue executing.
Format: CPH
Example:
```
SPB x250 'Set breakpoint on program 2 at step 5
EXF 2 'Start program 2
RDUMP 'Examine some parameter
SPB x210 'Set new breakpoint at step 16
CPH 'Allow program to continue
```

Explanation of Commands, Functions, Statements & Operators
(In Alphabetical Order)
(Continued)

CTRL-C
(Keyboard Execution only)

Definition: Causes a break in a presently executing program to occur, unless BOFF has been executed.
Format: press Ctrl-C
Example: A program is being executed that produces a number of information statements, only one of which is presently desired. After the desired statement has occurred, a CTRL-C will cause the program to stop execution.

DATA
(Statement)

Definition: Stores the numeric and string constants that are accessed by the program's READ statement.
Format: DATA constant, constant, . . .
The constants may be either numeric or string, and do not need to be surrounded by quotes unless the string contains commas, colons or spaces. The comma is the delimiter of the DATA statement and the colon, return or double quotation mark (") marks the end of a line.
Example: DATA 56, Eighteen, You have just WON, "1,000,000"

DAY$
(Function)
Definition: Returns the calendar day in the form MM-DD-YYYY
Format: DAY$(0)
The returned date is in the form MM-DD-YYYY where:
MM = the numeric representation for the month
DD = the numeric representation for the day
YYYY = the numeric representation for the year
Example: If the date is March 12, 1989, then Day$(0) will return: 03-12-1989.

DBA
(Statement)
Definition: Creates a file of the number of records and fields per record as specified.
Form: DBA file, records, fields
Example: 10 DBA 0,50,15 ‘ creates file 0 with 50 records, containing 15 fields.

Explanation of Commands, Functions, Statements & Operators
(In Alphabetical Order)
(Continued)

DBC
(Statement)
Definition: Copies the record buffer of one file into the record buffer of another.
Form: DBC file 1, File 2
Example: 100 DBC 1, 2 ‘copy file 1 record buffer into file 2

DBE
(Function)
Definition: returns the last error that occurred on the specified file from a BASIC operation
Format: X=DBE(n)
Example: 10 Print DBE(1) ‘ print the last error from file 1

DBF
(Function)
Definition: returns the number of fields per record in the specified file
Format: X=DBF(2)
Example: 10 X=DBF(2)

DBL
(Function)
Definition : Returns the current record position in the specified file
Format: L=DBL(n)
Example: 20 L=DBL(3) ‘find the current record in file 3

DBR
(Function)
Definition: Returns the number of records in the specified file
Format: R=DBR(n)
Example: 30 R=DBR(0) ‘prints how many record are in file 0

Explanation of Commands, Functions, Statements & Operators
(In Alphabetical Order)
(Continued)
DIM
(Statement)

Definition: Specifies the maximum values for array variable subscripts and allocates storage space.
Format: DIM variable(X, Y)
Variable is the array, and x and y are the values that define the array.
Example: DIM L1(29) to create a one dimensional array with 30 storage elements.

DIM B4(2,3) to create a two dimensional array with 12 storage elements.

Explanation of Commands, Functions, Statements & Operators
(In Alphabetical Order)
(Continued)

DPARM
(Function)
Definition: Returns the data value of the specified field in the record buffer of the specified file.
Format: A=DPARM(file, field)
Example: 10 A=DPARM (1,2) ‘Sets A= to the field 2 of file 1's record buffer.

DSET
(Statement)
Definition: Stores the data value into the specified field of the record buffer of the specified file.
Format: DSET file, field, data
Example: 10 F=2
20 D=500
30 DSET 1, F, D ‘Stores 500 into field 2 of file 1's record buffer.

Explanation of Commands, Functions, Statements & Operators
(In Alphabetical Order)
(Continued)

ECHK
(Function)

Definition: Returns the status of the reply to a message sent out the Events Port with ESEND. The status is one of four values:
0 = not ready
3 = time-out (no reply received)
6 = good reply received
15 = error code received (or bad checksum)
Format: Y=ECHK(0)
Example: 30 IF ECHK(0)=0 THEN 30 :REM. wait for reply

EDIT
(Command)

Definition: Displays a line, on the terminal, for editing.
Format: EDIT lll lll is the line number to be edited.
Ctrl-S moves the cursor to the left, and Crtl-D moves it to the right. Any key typed is inserted in the line at the cursor.
DEL, on the keyboard, deletes the character at the cursor, and backspace (Ctrl-H) deletes the character to the left of the cursor.
Esc, on the keyboard, aborts the editing process.
Return, on the keyboard, saves the line and ends the editing process.
Example: EDIT 120
This allows the editing of line 120 and causes it to appear on the screen.

Explanation of Commands, Functions, Statements & Operators
(In Alphabetical Order)
(Continued)

EOF
(Function)

Definition: Returns a zero if a complete message has been received thru the communications port; otherwise, returns a 1.
Format: Y=EOF(0)
An error occurs if the communications port is not open.
Example: 20 IF EOF(0)=0 THEN REC

EREC(0)
(Function)

Definition: Returns the last reply to an ESEND. If ECHK is 0 or 3, the reply will not relate to the message sent.
Format: A$=EREC(0)
Example: 10 ESEND A$
20 IF ECHR=0 THEN 20
30 IF ECHR=3 THEN 100:REM Time-out Error
40 A$=EREC(0)

ERR$
(Function)

Definition: Returns a string which contains the line number and the error.
Format: Y$=ERR$(0)
Example: 100 Y$=ERR$(0)
110 ? Y$
This will print the <line number where the error occurred> and <the error message>(i.e.: <80><OUT OF DATA>).

Explanation of Commands, Functions, Statements & Operators
(In Alphabetical Order)
(Continued)

ESEND
(Function)

Definition: Sends a message out the Events Port after computing the required checksum and adding the carriage return. Since the normal instrument Events communication continues, there is a slight delay before the message is sent. (See ECHK)
Format: ESEND A$
Example: 10 A$=?"01M"
20 ESEND A$

EXB
(Statement)

Definition: Begins execution of an instrument background program.
Format: EXB n
n is the program number or the program number plus 256 times the step number. If the step is 0, the program starts at step
Example: EXB 10 'Starts program 5 at step 4 or
EXB 4*256+5 'Starts program 5 at step 5
or
EXB x302 'Starts program 2 at step 3

Explanation of Commands, Functions, Statements & Operators
(In Alphabetical Order)
(Continued)

EXF
(Statement)

Definition: Begins execution of an instrument foreground program.
Format: EXF n ,n is the program number of the program number plus 256 times the step number. If the step is 0, the program starts at step 1.
Example: EXF 10 'Starts program 10 at step 1
or
EXF 4*256+5 'Starts program 5 at step 4
or
EXF x302 'Starts program 2 at step 3

EXP
(Function)

Definition: Calculates the exponential function. The exponential function is the number e raised to the power x.
Format: Y=EXP(x) x may be any numeric expression less than 88.02969
Example: Y=EXP(x)

for x=	Y=
0	1
-50	1.9287E-22
50	5.1847E21

Explanation of Commands, Functions, Statements & Operators
(In Alphabetical Order)
(Continued)

FIND

(Statement)

Definition: Searches the specified file for the value in the specified field. A condition code masked starting record must also be provided. The condition codes are

- = (equal)
- <(less than)
- >(greater than)
- n (not equal)

The mask is used for BIT or BYTE manipulation and must be Hex FFFF for numeric searches.

Format: FIND file, field, value, condition, mask, record.

Example:
```
10 Find 1, 5, 1000, >, xFFFF, 8.
20 Print "Found at"; DBL(1)
```
The above example searches file 1 field 5 for the first occurrence after record 8 of a value greater than 1000.

FOR
(Statement)

Definition: Creates a loop to perform a set of operations a given number of times.

Format:
```
FOR var=<expr> to <expr>(STEP X)
.
.
Next var
```
var is the variable used as the counter. expr are expressions used to define the range in which the loop is performed (IE: 0-50, or X=3*Y to X=Z-5). STEP x is an optional command that specifies what the counting increment should be. The counter is automatically incremented by 1 if no STEP command is given.

The operations that are performed are those located between the FOR statement and the NEXT statement, sometimes called a FOR...NEXT loop. FOR...NEXT loops may be nested with one another.

Example:
```
10 L=5:W=3
20 FOR N=1 TO L
30 PRINT W+N
40 NEXT N
```

Explanation of Commands, Functions, Statements & Operators
(In Alphabetical Order)
(Continued)

FRE
(Function)

Definition: Returns the amount of available, or "free" memory.
Format: Y=FRE(0)
Example: PRINT FRE(0)

FREE
(Statement)

Definition: Prints the amount of available, or "free" memory.
Format: FREE
Example: FREE
The amount of available memory will automatically be printed.

FST
(Function)
Definition: Reads the status of the specified file; 0 for a valid file or 1 for a not valid (undefined) file.
Format: A=FST (file)
Example: 10 If FST (0) = 0 then 100
20 Print “File not defined”

GET$
(Function)
Definition: Get a specified number of characters from the communications port. The port must have been previously opened before this function can be properly used.
Format: A$=GET$(n)
n = the number of characters to "GET". If there are not n characters in the buffer, then the number of characters available will be returned (minimum of 1). Waits if return is 0.
Example: 10 OPEN 1,4
20 SDLRC "1Ai": REM..... Request Setpoint
30 IF EOF(0)=1 THEN 30: REM..... Wait for Reply
40 A$=GET$(LOC(0)): REM.... Get all Characters

Explanation of Commands, Functions, Statements & Operators
(In Alphabetical Order)
(Continued)

GET
(Statement)

Definition: Reads the specified record into the record buffer of the specified file.
Format: GET file, record
Example: 10 GET 2,23 ‘get record 23 of file 2.

GOSUB
(Statement)

Definition: Calls a routine that is located at the given line number.
Format: GOSUB lll
.
.

lll is the line number at which the subroutine begins.
The RETURN statement causes the program to branch back to the line immediately following the GOSUB statement, and is located within the subroutine.

GOTO
(Statement)

Definition: Branches, unconditionally, from the normal program sequence to the specified line number.
Format: GOTO lll
lll is the line number to which the program branches.
Unlike the GOSUB, the program does not return to the line following the GOTO.
Example:
```
10 X=15
20 FOR I=1 TO 5000
30 Y=5*X+EXP(I*X)
40 IF Y<>83, THEN 60
50 GOTO 70
60 NEXT
70 PRINT X; Y; 165535.
```

Explanation of Commands, Functions, Statements & Operators
(In Alphabetical Order)
(Continued)

HEX$
(Function)

Definition: Returns a string which represents the hexadecimal value of the decimal argument.
Format: Y$=HEX$(x)
x is a decimal number in the range 0 to
Example: Y$=HEX$(x)

for x=	Y$=
256	x100
50	x32
672	x2A0
10000	x2710

Explanation of Commands, Functions, Statements & Operators
(In Alphabetical Order)
(Continued)

IF
(Statement)

Definition: Makes a decision regarding program flow based on the result of an expression.

Format: IF expr THEN action (ELSE action)

expr may be any numeric expression or operation. action is the steps that the program needs to perform next, it can be a BASIC command, or a line number that the program should branch to. The optional ELSE can be used to define action that should be taken if the expr is false. Multiple actions may be performed. The logical AND and OR can be used to combine expressions for multiple conditions with the following format:

IF expr 1 AND expr 2 THEN action

The order of the keywords AND and OR is established by placement and cannot be altered by parenthesis. Any number of AND's/OR's can be used, in any order, in the expression. The only requirement is that the entire IF...THEN..ELSE statement fit within one 80 character line.

NOTE: that the AND and OR are not the same as the logical operators & and @.

Example:

1. 10 IF X+Y>=50 THEN
   20 IF Y=Z+50 THEN Y=100:
   GOTO 200 ELSE GOTO 100
2. 10 IF A&B=C AND A$=C$ OR D@E=F THEN C=COS©

   In the above example, A&B=C is expr 1, A$=C$ is expr 2, and D@E=F is expr 3. The logical key words (AND and OR) are evaluated in the order encountered. Therefore, expr 1, whether true or false, is ANDed with expr 2, whether true or false and the new ANDed expression is true only if both expr 1 and expr 2 are true. This result is then ORed with expr 3, whether true or false, and the new expression will be true if either the ANDed expression, or expr 3 is true.

Explanation of Commands, Functions, Statements & Operators
(In Alphabetical Order)
(Continued)

INCHK(0)
(Function)

Definition: Returns a status byte of operator input operation. Bits 4 and 5 indicate which input mode the instrument is in (10H for logic, 20H for integer, and 30H for float). Bits 6 and 7 indicate a key pressed. Bit 7 only (80H) equals [Enter] key pressed. Bit 6 only (40H) equals [Shift] key pressed. Bits 6 and 7 (60H) equals [Setpt] key pressed. Used with the INGET(0) function.

Format: X=INCHK(0)

Example: 10 X=INCHK(0)

INFLOAT
(Statement)

Definition: Initiates the operator input mode for numbers within the range of ±0.000001 to 999999. The instrument will only switch
operator input mode if the instrument was in normal mode (IE: operator was not using the keyboard).

Format: INFLOAT m$,n m$ is the message to be displayed.
n is the default, or old value.
The float mode displays the first eight characters of the messages on one paragraph (four top, four bottom), and the value of n on another paragraph. The two paragraphs are displayed alternately for 2 seconds each until the operator presses a key. When the operator presses a key, the display will hold for 25 seconds. The value displayed has a d in the top display, followed by the sign of the value, and the first two numbers. The lower display will then show the remaining numbers.
The [Left Arrow] and [Right Arrow] are used by the operator to select the digit to change, and then changes the digit with the [Up Arrow] and [Down Arrow] keys. Selecting the "d" will allow the position of the decimal point to be changed with the [Up Arrow] and [Down Arrow] keys.

Example: 10 INFLOAT "NEW DATA", +1.23456

The value and message will be displayed as follows (alternating every 2 seconds): d+1.2NEW
3456DATA

Explanation of Commands, Functions, Statements & Operators
(In Alphabetical Order)
(Continued)

INGET(0)
(Function)

Definition: Returns the current value of the operator input. The value returned is always the value on display, even if the operator has not pressed Enter.
Format: X=INGET(0)
Example: 10 X=INGET(0)

ININT
(Statement)

Definition: Initiates the operator input mode for numbers within the range of 0 to 9999. The instrument will only switch to operator input mode if the instrument was in normal mode (IE: operator was not using the keyboard).
Format: INIT m$,n
m$ is the message to be displayed. n is the default, or old value.
The integer mode displays the first four characters to the message on the top display, and the integer value of the number in the bottom display. The [Left Arrow] and [Right Arrow] keys select thedigit to be changed, the [Up Arrow] and [Down Arrow] keys change the digit.
Example: 10 ININT "AGE",32
The instrument display will be as follows:

AGE
32

Explanation of Commands, Functions, Statements & Operators
(In Alphabetical Order)
(Continued)

INKEY$
(Function)

Definition: Returns a one character string equal to the last key pressed at the terminal.
Format: Y$=INKEY$(0)
INKEY$ will only read a single character, even if several have been pressed. If no key has been pressed, then a null string is returned ("").
Example: If a pause in the program is desired in order that setup procedures may take place, then the following could be used:

```
.
60 PRINT "Press any key to continue"
70 A$=INKEY$(0)
80 IF A$ ("")THEN 70
.
```

INLOGIC
(Statement)

Definition: Initiates the operator input mode for yes or no responses. The instrument will only switch to operator mode if the instrument was in normal modeIEie: operator was not using the keyboard).
Format: INLOGIC m$, n
m$ is the message to be displayed.
n is the default, or old value.
The logic mode displays the first four characters of the message on the top display, and a "Y" (if n is nonzero) or a "N" (if n is zero) in the bottom display. The [Up Arrow] and [Down Arrow] keys toggle between the yes and no response.
Example: 10 INLOGIC "OK", 1
The instrument display will be as follows:
OK
Y

Explanation of Commands, Functions, Statements & Operators
(In Alphabetical Order)
(Continued)

INPUT
(Statement)

Definition: Receives input from the keyboard and stores it as the given variable.
Format: INPUT var or input A$
var is the variable name under which the input information is stored. A$ is the variable name under which input text information, quotation marks removed, and no spaces allowed is stored.
Example: 10 INPUT X
20 PRINT X "Squared is" X^2

INPUT LINE
(Statement)

Definition: Receives input from the keyboard and stores it as the given variable.
Format: INPUT LINE A$
A$ is the variable name under which input text information is stored. INPUT LINE A$ allows for embedded spaces and quotation marks.
Example: 10 INPUT LINE X$
20 PRINT X$ "has just WON..."

INQUIT(0)
(Statement)

Definition: Cancels the operator input mode, and returns to the normal mode.
Format: INQUIT
Example: 10 INQUIT

Explanation of Commands, Functions, Statements & Operators
(In Alphabetical Order)
(Continued)

INSTR
(Function)

Definition: Searches for a given string within another given string, beginning at a specified position. The beginning position of the desired string is then returned.

Format: Y=INSTR (xA$,B$)
B$ is the string to be searched for. A$ is the string that is searched. x is the position in A$ at which the search begins. If B$ is not found, then 0 is returned. x must be specified.

Example:
```
10 A$="checkout 57"
20 Y=INSTR (x, A$,B$)
```

for x=	for B$=	Y=
1	e	3
5	e	0
1	c	1
3	c	4

INT
(Function)

Definition: Returns the largest integer that is less than or equal to x.

Format: Y=INT(x)
x is any numeric expression.

Example:
```
Y=INT(x)
```

for x=	Y=
-1.1	-2
1.1	1
-3.68	-4
3.68	3

Explanation of Commands, Functions, Statements & Operators
(In Alphabetical Order)
(Continued)

LEFT$
(Function)

Definition: Returns a character string from the first character position to the specified character position.

Format: Y$=LEFT$(A$,n)
A$ is the specified string.
n is the character position at which the return will end.

Example:
```
10 A$="checkout 57"
20 Y$=LEFT$(A$,n)
```

for n=	Y$=
4	chec
7	checkou
2	ch

LEN
(Function)

Definition: Returns the number of characters the specified string.

Format: Y=LEN(A$)
A$ is the specified string whose character length is stored in Y.

Example: Y=LEN(A$)

for A$=	Y=
check	5
Unipro	6
hi	2

LET
(Statement)

Definition: Assigns the value of an expression to a variable.

Format: LET var=exp
var is the variable name to which the expression is to be assigned.
exp is the expression that is to be assigned to the variable.
When assigning an expression to a variable name, the word LET is optional.

Example: LET Y=5*30+X^4 or Y=5*30+X^4

Explanation of Commands, Functions, Statements & Operators
(In Alphabetical Order)
(Continued)

LIST
(Command)

Definition: Lists the current program on the screen or other specified devices.

Format: LIST (lll) (-yyy)
lll is the line number to be listed if a single line is desired; or, it is the first line to be listed in a set of lines being listed.
yyy is the ending line to be listed is a set of lines are desired.
If no line number, or numbers, follows the LIST command, then the whole program will be listed.

Example: LIST 55
This command will list line 55.
LIST 40-80
This command will list all program lines between 40 and 80.
LIST
This command will list the whole program.

LOC
(Function)

Definition: Returns the number of characters received into the communications buffer.

Format: Y=LOC(0)
An error will occur if the port is not opened.

Example: 10 IF LOC(0)<>0 THEN BFCLR

LOG
(Function)

Definition: Returns the logarithm to the base 10 of x.

Format: Y=LOG(x)

Example: Y=LOG(x)

for x=	Y=
10	1
36	1.556
50	1.699

Explanation of Commands, Functions, Statements & Operators
(In Alphabetical Order)
(Continued)

LN
(Function)

Definition: Returns the natural logarithm of x. The natural logarithm is the logarithm to the base e.

Format: Y=LN(x)
x is a numeric expression greater than zero.

Example: Y=LN(x)

for x=	Y=
2.7183	1.00
10	2.3026
56	4.0025

Explanation of Commands, Functions, Statements & Operators
(In Alphabetical Order)
(Continued)

MGET
(Function)

Definition: Retrieves message packet from the BASIC port via a special data transfer command.

Message pack

format: sOBBEE[dddd]CC

s Start of message character, always a less than (<) sign.

O Operation character, the software supports the following two characters: R-Read process data (reads data values for message words from Multipro memory) and W-Write process data (writes data values from message words to Multipro memory).

BB Beginning word, the word specifying the address of the first word of data (all words are 16 bit, 2 bytes).

EE Ending word, the word specifying the address of the last word of data.

dddd Optional data field, length and type is dependent on operation character.

CC Checksum, is the 16 bit sum of the operation character, and all words in the "BB", "EE", and "dddd" fields.

Note: MGET along with MPROC perform very special functions relating solely to communication initiated by external devices through the BASIC port. For more specific application information, contact

Format: X=MGET(ttt)

ttt=the time-out value in hundreds of seconds, with a maximum of 250.

If X=0, Process OK

If X=3, Time-out Error

If X=21, Proper command character not found

Example:
```
100 X=MGET(100):IF X=3 THEN GOTO 100
110 IF X=0 THEN X+MPROC(0)
120 GOTO 100
```
Notes: You must set BOFF prior to using MGET.

Explanation of Commands, Functions, Statements & Operators
(In Alphabetical Order)
(Continued)

MID$
(Function)

Definition: Returns the requested part of a string.

Format: Y$=MID$(A$,C, n)
A$ is a string expression.
C Is the numeric character position at which the new string is to begin.
n is the length of the new string.

Example:
```
10 A$=Bob Ted Chris Bill
20 Y$=MID$(A$,C, n)
```

For C=	For n=	Y$=
1	4	Bob
6	7	ed Chri
17	4	Bill

MPROC
(Function)

Definition: Processes information retrieved by the MGET function. MPROC will either send instrument parameter values o the BASIC port or store values received from the BASIC port into instrument parameters, based on what is asked for in the message packet retrieved by MGET.

Format: X=MPROC(0)
If X=0, Process OK
If X=3, Time-out Error
If X=21, Proper command characters not found

Example:
```
100 X=MGET(100):IF X=3 THEN GOTO 100
110 IF X=0 THEN X=MPROC(0)
120 GOTO 100
```

Explanation of Commands, Functions, Statements & Operators
(In Alphabetical Order)
(Continued)

NEXT
(Statement)
Definition: Instructs an active FOR loop to increment the counter and return to the beginning of the loop.
Format: FOR var=<expr> to <expr>
.
NEXT var
The var after the NEXT is required.
Example: 10 L=5:W=3
20 FOR N=1 to L
30 PRINT W+N
40 NEXT N

NEW
(Command)
Definition: Initializes BASIC and clears all variables and pointers. The command NEW* initializes BASIC, but retains variable values.
Format: NEW or NEW*
Example: It is desired to start work on a new program after completing the present one. After all changes have been saved, the command of NEW, or NEW*, at the prompt will clear all memory and provide a new working area.

NUM$
(Function)
Definition: Converts the number x to a string with a leading space, or a - sign.
Format: NUM$(x)
x is a numeric expression.
Example: 10 A$=NUM$(100)
20 ? A$,LEN(A$)
RUN
100 4

Explanation of Commands, Functions, Statements & Operators
(In Alphabetical Order)
(Continued)

ON...GOTO
(Statement)
Definition: Control program flow to a subroutine based on the value of a numeric expression.When the subroutine is finished, it returns to the line following the ON...GOSUB. All other requirements are the same as the ON...GOTO.
Format: ON x GOTO l1, l2, l3, l4, l5, l6, l7, l8, l9, l10

x is a numeric expression.
l1 through l10 are line numbers to which control is passed if the integer value of x I s1 thru 10 respectively. Line numbers must exist in the program and are separated by commas. It is not necessary to include all ten line number; however, no line numbers can be skipped.

Example: 10 ON Y/2 GOTO 110, 120, 130, 140
20 GOSUB 640
If Y/2=1 then program execution goes to line 110.
If Y/2=2, execution goes to line 120, =3 goes to 130, and =4 goes to 140. Otherwise execution continues at the next line (line 20 in this example).

ON...GOSUB
(Statement)

Definition: Control program flow to a subroutine based on the value of a numeric expression. When the subroutine is finished, it returns to the line following the ON...GOSUB. All other requirements are the same as the ON...GOTO.

Format: ON x GOSUB l1, l2, l3, l4, l5, l6, l7, l8, l9, l10
x is a numeric expression.
l1 thru l10 are line numbers of the subroutines.

Example: 10 INPUT "Your Selection":A
20 ON A GOSUB 100,200,300,400
30 REM. Subroutines return here

Explanation of Commands, Functions, Statements & Operators
(In Alphabetical Order)
(Continued)

ONERROR
(Statement)

Definition: Defines the line number to which program control is sent should an error occur.

Format: ONERROR lll
lll is the line number to which program control is sent.
ONERROR without a line number will cancel the error trapping, thereby causing errors to stop program execution.

Example: ONERROR 120

This sends program control to line 120 should an error occur.

OPEN
(Statement)

Definition: Opens the communications port for BASIC to use.

Format: OPEN x, y
x is the mode.
y is the ASCII value of the end of the message character.
Mode selects half or full duplex, 7 bit even parity, or 8 bit no parity, and the BAUD rate. Mode is a number from 0-15 that can be calculated in the following manner:

x=D+P+B

D=0 for half duplex, or 8 for full duplex.

P=0 for 7 bit even parity, or 4 for 8 bit no parity.

B=0 for 1200 BAUD, 1 for 4800 BAUD, 2 for 19.2K BAUD, 3 for 76.8K BAUD, 16 for 2400 BAUD, 17 for 9600 BAUD, or 18 for 38.4K BAUD.

The end of message character can be any valid ASCII character. Normally this would be a 13 (carriage return) for normal terminal operation, or a 4 (EOT) for instrument communications. The end of message character must be a 4 to use the SDLRC statement. If the end of message character is 255 then it is ignored.

Example: 20 OPEN 3,4

This opens the communications port for half duplex, even parity, 4800 BAUD with an EOT as the end of the message character.

Explanation of Commands, Functions, Statements & Operators
(In Alphabetical Order)
(Continued)

@ (OR)
(Logical Operator)

Definition: Logical operation used for bit by bit comparison. If two bits in the same position are both 0's, then the result will be 0. If, however, one or both bits are 1, then the result will be a 1.

Format: <exp>@<exp>
exp are numeric.

Example: Binary: 0010 0111 0001 0000
@ 0000 0001 0010 1100
= 0010 0111 0011 1100
Decimal: 10000 @ 300 = 10044
Hex: x2710 @ x12C = x273C

All three of the above examples have same numerical value, but only the binary demonstrates the actual process which determines the outcome of the @ operator.

Notes: The keyword OR is used in IF...THEN statements to logically OR conditions. See the IF...THEN statement for more details.

PMO$
(Function)

Definition: Displays a string in the instrument page display. When executed, the command forces the page display to be displayed.

Format: B$=PMO$(A$)
A$ is a string from which the 8 farthest left characters are available on the instrument page display.
If the string is less than 8 characters, it is padded with spaces
Decimal points are merged with other characters, and therefore for do not count as a character themselves.

Example:
```
10 A$="1086+45.3"
20 B$=PMO$(A$)
```
The message on the instrument display page will contain 1086 on the top and +45.3 on the bottom. The whole message is returned because the "." will be merged with the 5 and will not require a space of it's own.

Explanation of Commands, Functions, Statements & Operators
(In Alphabetical Order)
(Continued)

PM1$
(Function)

Definition: Displays a string on the instrument page display. When executed, the command forces the page display to be displayed.

Format: B$=PM1$(A$)
Behaves the same as that of PMO$ except that it has its own location in the instrument display page (IE: two individual messages can be displayed in the instrument at the same time, and can be viewed by simply changing the display page).

Example: See PMO$

PM2$
(Function)

Definition: Updates the string displayed in the operator input mode, allowing the BASIC program to scroll or otherwise change the message. Unlike PMO$ and PM1$, PM2$ does not automatically force the instrument display.

Format: PM2$(A$)
A$ is the string message to be displayed.

Example: See PMO$.

PARM
(Function)

Definition: Returns the instrument parameter specified.

Format: A=PARM(num)
num is normally 0 to 239 and applies to the value of the local instrument.
If the instrument is a network master, then a slave instrument can be accessed by adding 256*CH addr to the parameter number. (CH addr is the channel address)
If the number is entered in hexadecimal, note that 100 (hex) = 256.

Example: A=PARM(x206)
This accesses parameter 6 of the second slave channel.

Explanation of Commands, Functions, Statements & Operators
(In Alphabetical Order)
(Continued)
PEDIT
(Command)
Definition: PEDIT allows access to the instrument program editor commands.
Format: PEDIT
The PEDIT commands are explained in the following:

1. nn OP DE SR Comment
   Enter a step by typing the step number (nn, space, the OPCODE (OP), space, the destination (DE), space and the source (SR).Step numbers can be 1 or 2 digits and must be between 1 and 24.
   OPCODE is a 2-digit valid programmer OPCODE.
   Destination is a 2-digit valid destination register, or when appropriate, a condition code.
   SouRce is the source entered in one of three ways. If specifying a register, then it must be a 2-digit valid register. If specifying decimal data, it must be entered with at least 3 digits (IE: enter 2 as 002). If specifying Hex data, then the first character must be a lower case x followed by 1 to 4 valid Hex digits.
   Comments following the SouRce are ignored.
   One space character after the SouRce indicates a comment. Comments are not stored in the instrument; however, programs downloaded from a computer may have comments.
2. CLR
   Clears the edit buffer so that all 24 steps are NOPs.
3. DUMP
   DUMP lists the program in a single column format. This command allows the program to be easily "captured" by a terminal emulator program on a computer and saved into disk.

Explanation of Commands, Functions, Statements & Operators
(In Alphabetical Order)
(Continued)

PEDIT
(Command)
Format (Continued)

4. EDIT nn
   Puts the step nn into the line edit buffer so that changes can be made. The editor is in insert mode; CTRL-S moves the cursor left, CTRL-D moves the cursor right. DEL deletes the character under the cursor, BS deletes the character to the left of the cursor, ESC aborts editing without saving the changes, and ENTER saves the changes.
5. END
   Returns to the BASIC interpreter.
6. HELP

Lists the PEDIT commands on the terminal with a short description.

7. LIST
   Displays the program in a two column format. Source data is displayed in both hexadecimal and decimal.
8. LOAD nn
   Loads the instrument program number nn into the editor for modification.
9. SAVE nn
10. Error Codes
    SN Step number is invalid
    OP OPCODE is invalid
    DE Destination is invalid
    SR Source is invalid

PEEK
(Function)

Definition: Returns the decimal value read from the indicated memory position.

Format: A=PEEK(addr)
addr is the memory position from which the information is to be read.

Example: A=PEEK(54)
This reads the numeric value (16-bit signed) from memory position 54.

Explanation of Commands, Functions, Statements & Operators
(In Alphabetical Order)
(Continued)

POKE
(Statement)
Definition: Stores information into the specified memory address.
Format: POKE n, m
n is the memory location to which the information is to be stored.
m is the information that is to be stored in that location
Example: POKE 32,100
This places information specified by the numeric value of 100 into location 32.

PRINT
(Statement)
Definition: Displays the specified data onto the screen.
Format: PRINT <exp>
exp is a list of numeric and or string expressions separated by commas, blanks, or semicolons.
All string constants must be enclosed within quotation marks.
? can be used instead of PRINT to obtain the same results.
If the PRINT, or ?, command is used with no expression, then a line feed is executed.
Example: 10 X=560
20 Y=5000
30 PRINT "If your number is "X", then you have just won "Y"
READ B$,A4
This reads the first available data from the DATA statement as a string variable, and the second as a numeric variable.

PUT
(Statement)
Definition: Writes the file buffer to the specified record of the file.
Format: PUT file, record
Example: 10 PUT 0,15 ‘writes the record buffer of file 0 to record 15.

Explanation of Commands, Functions, Statements & Operators
(In Alphabetical Order)
(Continued)

REC
(Statement)

Definition: Reads the received message from the communications buffer and stores it as the given string variable.

Format: REC A$

A$ is the variable to which the received information is stored.
An error occurs if the communications port is not opened.
Do not use REC if the end of message character in the open statement is set to 255.

Example:
```
10 OPEN 3,13
20 SEND "Hello"+CHR$(13)
30 REC A$
```

REM.
(Statement)

Definition: Allows for the insertion of explanatory remarks in a program.

Format: REM. remark

or

'remark

remark may be any sequence of characters.
REM... statements are not executed and, therefore, do not affect the executing of the program or its variables.

Example:
```
10 REM.
this program is designed to output the area.
```

RESTORE
(Statement)

Definition: Allows DATA statements to be reread from the beginning of the program.

Format: RESTORE

Example:
```
10 READ A1, B1, C
20 RESTORE
30 READ D1, E1, F
```
This allows for the same information to be read into variable D1 as was read into A1, and the same information to be read into E1 as was read into B1, etc.

Explanation of Commands, Functions, Statements & Operators
(In Alphabetical Order)
(Continued)

RETURN
(Statement)

Definition: Used in conjunction with a GOSUB statement, this program returns program operation from a subroutine back to the main program.

Format: GOSUB lll
.
.
RETURN
lll is the line number location of the subroutine.

Example:
```
100 GOSUB 200: X=4
110 ? X
120 END
200 RETURN
```

RIGHT$
(Function)

Definition: Returns a string from a designated number of characters at the end of a specified string.

Format: Y$=RIGHT$(A$,n)
A$ is the string function from which the new string will be formed.
n is the number of characters from the end of A$.

Example:
```
10 A$=·"checkout 57"
20 Y$=RIGHT$(A$,n)
```

for n=	Y$=
8	eckout57
4	ut57
1	7

Explanation of Commands, Functions, Statements & Operators
(In Alphabetical Order)
(Continued)

RND
(Function)

Definition: Generates a random number between 0 and X.

Format: RND(X)
X is a numeric expression used as the upper limit of the random number generation.
If X is omitted, a number is generated between 0 and 1.

Example: RND(40)
This statement will produce a random number between 0 and 40.
Y=RND(40)
This statement will store the number produced by a random number generation

between 0 and 40 in the variable Y.

RUN
(Command)
Definition: Begins execution of the current program.
Format: RUN
Example: RUN

SDLC
(Statement)
Definition: Adds a delimiter, LRC, and EOT to a string and transmits it out the communications PORT.
Format: SDLRC A$
This statement is used when communicating with MSI or Barber Colman instruments. An error occurs if the communications port is not open.
Example: 10 OPEN 3,4
20 S=1200:REM desired setpoint
30 A$="1Al"+NUM$(S)
40 SDLRC A$

Explanation of Commands, Functions, Statements & Operators
(In Alphabetical Order)
(Continued)
SEND
(Statement)
Definition: Transmits a string out the communications port.
Format: SEND A$
An error occurs if the communications port is not open.
Example: 10 OPEN 3,13
20 SEND "HELLO"+CHR$(13)
30 REC A$

SETPAR
(Statement)
Definition: Sets an instrument parameter to the specified value.
Format: SETPAR num, value
num is usually between 0 and 239 and represents the value of the local instrument.
value represents the value to which the specified instrument parameter is to be set.
If the instrument is a network master, then a slave instrument can be accessed by adding 256*_CH addr to the parameter number (CH addr is the channel address).
If the number is entered in hexadecimal, note that 100 (hex)=256.
Example: SETPAR x206,50
This sets parameter 6 of the second slave instrument to 50.

SGN

(Function)

Definition: Returns the sign of x.

Format: Y=SGN(x)
x is any numeric expression.
If x>0 then SGN x=1. If x<0 then SGN x=-1. If x=0 then SGN x=0.

Example: Y=SGN(x)

for x=	Y=
50	1
-50	-1
0	0

Explanation of Commands, Functions, Statements & Operators
(In Alphabetical Order)
(Continued)

SIN
(Function)

Definition: A mathematical function that returns the sine of an angle given in radians.

Format: Y=SIN(x)
x is an angle given in radians.
Degrees may be converted to radians by multiplying the degrees by p/180. (p is equal to 3.14159)

Example: Y=SIN(x)

for x=	Y=
PI/3	.8660
3PI/4	.7071

SIO
(Statement)

Definition: Causes the terminal UART to be reset to 9600 BAUD, no parity, 8 bits. Used if setup values are being lost or need to be checked.

Format: SIO

SLEEP
(Statement)

Definition: Used in either a program or as a single command to place all instruments, whose BASIC ports are connected, to sleep. A single instrument may then be activated with the AWAKE command, or the @ symbol may be used to issue a command to all of the instruments. When used in conjunction with the AWAKE statement, multi-dropped instrument communication becomes more convenient.

Format: SLEEP

Note: An @RUN will start a program in every instrument, or an @ may be placed in front of each command of a program, allowing for the same program to be downloaded to each instrument simultaneously.

Example:
```
10 SLEEP
20 @RUN
```

Notes: When in the sleep mode, BASIC will accept any characters thru its port. If a program is running and the instrument is asleep, any print commands given in the program will be discarded, and not transmitted.

The @ symbol at the beginning of a line will allow BASIC to receive that line if the instrument is asleep.

Explanation of Commands, Functions, Statements & Operators
(In Alphabetical Order)
(Continued)

SLSTAT(0)
(Function)

Definition: Returns the sleep or awake status of an instrument. This command is used in a program in the instrument in order to determine if communications to the terminal is possible.

Format: X=SLSTAT(0)
If the instrument is awake, a 0 is returned.
If the instrument is in sleep mode, then a 1 is returned.

Example:
```
10 X=SLSTAT(0)
20 IF X=0 THEN PRINT "Hello":REM. only print if awake
```

SPACE$
(Function)

Definition: Returns a string with a specified number of space characters.

Format: Y$=SPACE$(n)
N is the number of spaces the string is to contain.

Example:
```
10 ?"HI",SPACE$(5),"THERE"
RUN
HI     THERE
```

SPB
(Statement)

Definition: Sets a program breakpoint at a specific program number and step for debugging the foreground program. When the breakpoint is reached, the program is placed in hold and the breakpoint cleared.

Format: SPB n
n is 256*program number + the step number of the breakpoint.

Example: SPB 10*256+7 'sets breakpoint at step 7 of program 10. SPB x0A07 'same as above using Hex data

Explanation of Commands, Functions, Statements & Operators
(In Alphabetical Order)
(Continued)

SQR
(Function)
Definition: Returns the mathematical square root of the specified number.
Format: Y=SQR(x)
x is a number greater than or equal to 0.
Example: Y=SQR(x)

for x=	Y=
4	2
81	9
53	7.28

SSP
(Statement)
Definition: Single steps a foreground program in order to assist in debugging the program.
Format: SSP
SSP will allow a program to hold to execute one step. If a program was not in hold, it will be placed in hold.
Example:
```
SSP 'set single step mode
EXF 10   'start program 10, only 1 step is executed
SSP 'next step is executed
```

STOP
(Statement)
Definition: Terminates program execution and returns to command mode.
Format: STOP
CONT will restart the program.

STRING$
(Function)
Definition: Returns a string with a specified number of ASCII characters whose decimal value is also given.
Format: Y$=STRING$(n, d)
n is the number of ASCII characters the string is to contain.
d is the decimal value of each character in the string.
Example:
```
10 B$=STRING$(4,42)
20 ? B$
RUN
****
```

Explanation of Commands, Functions, Statements & Operators
(In Alphabetical Order)
(Continued)

TAN
(Function)
Definition: A mathematical function that returns the tangent of an angle specified in radians.
Format: Y=TAN(x)
x is the angle in radians.
Degrees may be converted to radians by multiplying the degrees by p/180. (p is equal to 3.14159)
Example: Y=TAN(x)

for x=	Y=
PI/3	1.732
3PI/4	-1.000

TIME$
(Function)
Definition: Returns the time of day in the form HH:MM:SS.S.
Format: B$=TIME$(0)
The time is returned in the form HH:MM:SS.S, where:
HH =the hour
MM=the minutes
SS.S=the seconds.
Example: If the time is precisely 5 after 11 the morning, then TIME$(0) will return: 11:05:00.0

VAL
(Function)
Definition: Converts a numeric string to a numeric value.
Format: Y=VAL(A$)
A$ is the numeric string to be converted to a numeric value. The string must contain only numbers.
Example: 10 A$="1000.0E3"
20 Y=VAL(A$)
Y will equal 1,000,000

Explanation of Commands, Functions, Statements & Operators
(In Alphabetical Order)
(Continued)

XEQ
(Command)
Definition: Executes the current program without zeroing the variables.
Format: XEQ
Example:
```
10 A=A+1
20 ? A
RUN
1
XEQ
2
XEQ
3
RUN
1
```

% (XOR)
(Logical Operator)
Definition: A logical operator used for bit by bit comparison. If two bits in the same position have the same value (IE: both are 1's or both are 0's), then the result is 0. If they have two different values, then the result is 1.
Format: <expression>%<expression>
The expressions are numeric.
Example: Binary: 0010 0111 0001 0000
% 0000 0001 0010 1100
= 0010 0110 0011 1100
Decimal: 10000 % 300 = 9788
Hex: x2710 % x12C = x263C
All three of the above examples have the same numerical value, but only the binary demonstrates the actual process which determines the outcome of the % operator.

BASIC ERROR TABLE

The following error message will be displayed when the running program encounters the described error.

Error Messages.	
Message	Description
UNDEFINED LINE	Program execution is transferred to a line number, which does not exist in the current program.
FILE NOT OPEN	If the Comm port is not open for send or receive
OUT OF MEMORY	A variable is assigned which is too large for available memory space.
MATH	Numbers used during math operations are too large.
OPEN DENIED	When the BASIC port is being used by another process and an open is attempted.
STACK	Temporary data stack is full.
SYNTAX	Improper usage or form.
TYPE MISMATCH	Improper usage of variable types (types do not match).
RETURN W/O GOSUB	RETURN statement found and no GOSUB was called to return from.
OUT OF DATA	A READ is attempted when no input exists or pointer is at the end of input data.
NEXT W/O FOR	NEXT used and no FOR statement is used.
CONVERSION	Wrong type of data used as input, cannot be converted.
OVERFLOW	Math calculation becomes too large for available memory.
UNDERFLOW	Math calculation becomes too small (negative), the exponent is to large.

Index