P&T-488 INTERFACE
INSTRUCTION MANUAL

MSOFT Software Package

PICKLES & TROUT
P. O. BOX 1206, GOLETA, CA 93116, (805) 685-4641

P&T-488 INTERFACE
INSTRUCTION MANUAL

Pickles & Trout
P.O. Box 1206
Goleta, CA 93116

WARRANTY

This Pickles & Trout product is warranted against defects in materials and workmanship for 90 days from the date of shipment. Pickles & Trout will, at its option, repair or replace products which prove to be defective within the warranty period provided they are returned to Pickles & Trout. Repairs necessitated by modification, alteration or misuse of this product are not covered by this warranty.

NO OTHER WARRANTIES ARE EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTIBILITY AND FITNESS FOR A PARTICULAR PURPOSE. PICKLES & TROUT IS NOT LIABLE FOR CONSEQUENTIAL DAMAGES.

FOREWORD

This manua contains the information necessary to understand and use the P&T-488 interface as well ; provide instruction in the basic concepts of the IEEE-488 bus.

Those who ; already familiar with the IEEE-488 bus (also known as the HP-IB, GPIB and ASCII bus) and he concepts of a Talker, Listener and Controller may skip to page MSOFT-1. It is recommende that those who are not acquainted with Talkers, Listeners and Controllers read the chapter "The IEEE-488 Bus" first.

The P&T-48: interface consists of two major components: the P&T-488 interface board and the P&T-488 tc Microsoft Basic interface software package. The software package is actually two programs: M: FT.COM and MSOFT.REL. MSOFT.COM is an object code program to be used with interp ter Basic, and MSOFT.REL is a linkable module to be used with compiler Basic. Also inc led is an object code program (488TST81) which performs a complete functional test of the P&T- 38 interface board. Additional programs are provided as examples of how one can use the P&T 88 interface to communicate with 488 devices. Several utility programs have also been provide . One which is especially useful is called BUSMON. It displays all IEEE-488 bus transactions the console and can also be used to send data or commands over the bus. All programs are ovided on a single density, non-system diskette recorded in CP/M† format.

Even though ISOFT is designed to be used with Microsoft Basic, other languages can also successfully use MSOFT. Included in this manual and on the disk are sample programs written in assembler, For an, Pascal and C.

† CP/M is a tr emark of Digital Research

Table of Contents

P&T-488 to Microsoft Basic Interface Software

(at its option) repair or replace them upon receipt of the defective pieces. Repairs necessitated by alteration, modification or misuse of these products are not covered by this warranty. Out of warranty interface boards which have not been modified or otherwise tampered with will be repaired or replaced for a flat fee. As of January, 1981, the fee is $45.00.

NOTICE - A handling fee of $45.00 will be charged for any board that is returned for repair because the wrong test routine was used. THIS INCLUDES BOARDS STILL IN WARRANTY.

When returning equipment to Pickles & Trout, be sure to include the following information:

1 NAME and ADDRESS of the owner.

2 NAME and PHONE NUMBER of the person who is using the P&T-488.

3 Description of the failure and how it was found. PRINTOUT OF THE TEST RESULTS IS REQUIRED.

4 Description of the S-100 machine and operating system. Include manufacturer and model name of the CPU board, system clock rate, and the name of the organization that authored the operating system, as well as any information on systemic modifications made to it.

For example: IMSAI 8080 with Ithaca Audio Z-80 CPU board with a system clock of 4 MHz, North Star single density 5.25" floppy disk drive and controller, Digital Research CP/M as modified by Lifeboat Associates for North Star disks.

5 If the equipment is still in warranty, enclose a copy of the bill of sale. Otherwise enclose a check for the repair and shipping and handling fees. The shipping and handling fee is $5.00 for addresses within the contiguous US, $7.50 for Alaska and Hawaii. There is no shipping fee for foreign addresses because the equipment will be returned freight collect.

The repairs/replacements will be made within five business days and the equipment returned postage paid to US addresses, freight collect to foreign addreses.

Appendices

Auxiliary Programs

- CAST OF CHARACTERS -

The 488 bus is populated by three major types of devices. One is the **Controller**, which sends commands over the bus to other devices. Another is the **Talker**, which sends data over the bus to one or more devices of the third kind: the **Listeners**. The Listeners and Talker communicate with a handshake on each data transfer, and the communication proceeds at the maximum rate allowed by the Talker and the slowest Listener. This communication is completely asynchronous and may be interrupted at specific points in the handshake cycle without causing any loss of data.

It can be useful to liken the bus to a meeting which has a chairman (Controller), a recognized speaker (Talker) and an audience (Listeners). As is true of most meetings, some of the audience is paying no attention whatever to the proceedings (some of the devices on the bus may be idle), while some of those that are listening want to interrupt the Talker. Sometimes a member of the audience is audacious enough to indicate that it should be the chairman. The 488 bus specification allows the Controller to designate another device as his successor.

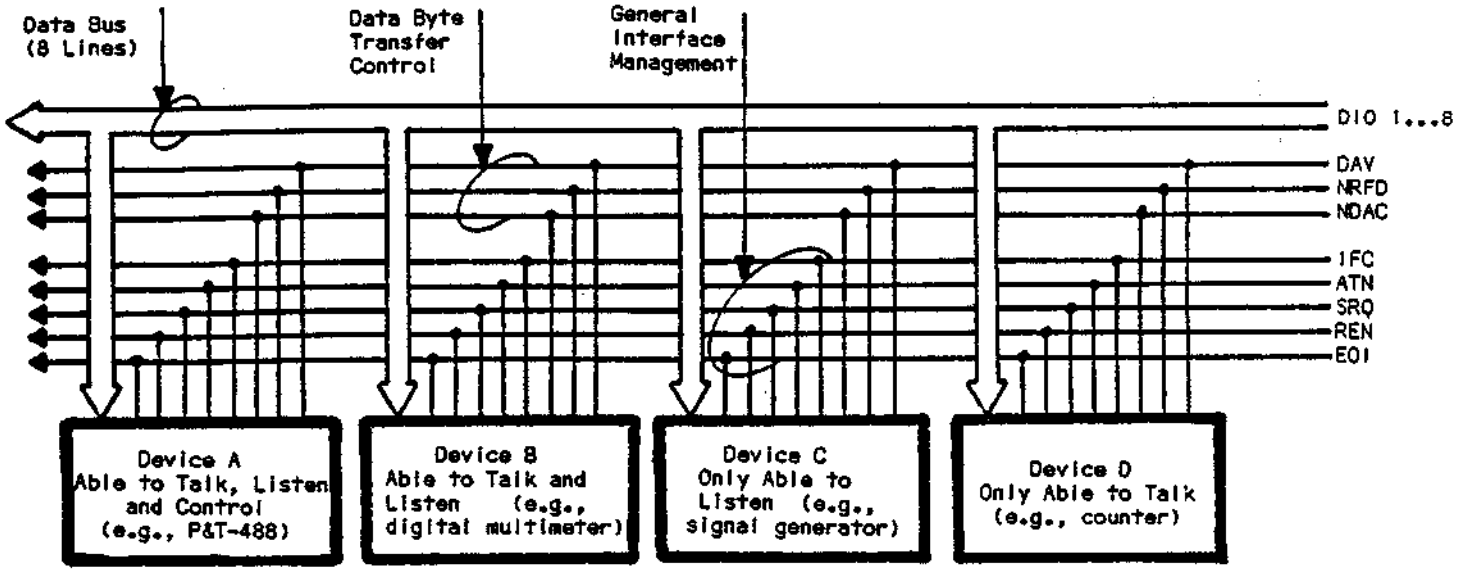
It is the **Controller's** responsibility to make sure that communication takes place in an orderly manner: it is he that says who can talk and who should listen at any given time. It is also the Controller that takes care of such matters as telling everyone to shut up (Universal Untalk **UNT** command), everyone to go back to their desks (Interface Clear **IFC**), or listen to someone trying to gain the floor (Service Request **SRQ**). Even though the Controller has (in theory) complete command over everyone else, problems can arise. One possible problem is that the Controller has made the unwise choice of telling more than one device that it can be a Talker, which results in sheer bedlam. Another way for the Controller to lose control of the situation is if a Talk Only (**ton**) device is placed on the bus. Some Talk Only devices are notoriously deaf and don't pay any attention to anybody, even the Controller!

A **Talker**, on the other hand, leads a simple life. It does not concern itself with disputes over who has the right to be heard, and when. It only puts data on the bus, waits until the slowest listener indicates it is ready for data, says the data is valid, waits until the slowest Listener says it has accepted the data, then says that it is removing the data and follows up on its threat. About the only thing that bothers a Talker is to find that no one is listening to him. Most get really upset and let the Controller know about this impolite state of affairs. Talkers that don't complain have a tendency to sit there with their mouths open, caught in mid-word. Either way, no communication is taking place and this is not considered a desirable state of affairs.

Listeners can be a little more complicated. They let the Talker know when they are ready for another word and when they have received it. Some also let the Controller know that they want some special attention. The Controller waits until the Talker can be interrupted so that no Listener is deprived of the latest bit of wisdom imparted by the Talker. Then the Controller tries to find out which device wants the attention. Two ways to do this are **Serial Poll**, in which each device is allowed to speak (one at a time) and **Parallel Poll**, which allows several devices to simultaneously inform the Controller of their need by a bit pattern each puts onto the eight data lines.

- HARDWARE OVERVIEW -

The 488 bus is made up of 16 signal lines: eight are used for data, three are needed for the interlocking handshake used to communicate the data, and the remaining five are used for bus management. Since there are eight data lines, a full eight bit byte can be communicated in each handshake cycle. This is what is meant by the phrase "bit parallel - byte serial" transmission. It is an alternative to the slower RS 232C standard, in which only one data line is used (and which is referred to as being a "bit serial" interface standard).



There are three basic concepts which are important to an understanding of how the hardware of the 488 bus works. The first is that only one of two voltages is allowed on each line, and the lower allowed voltage is ground. The second is that the 488 bus uses negative true logic, which means that the lower of the two voltage levels has the value TRUE, while the higher voltage has the value FALSE. The third is that the bus uses open-collector drivers. An open-collector driver can be thought of as a switch with one terminal connected to the line and the other to ground. When the driver is ON, it is as if the switch is closed, and so connects the line to ground. If the driver is OFF, it is as if the switch is open, so no connection is made between the line and ground. There is a resistor connecting the line to a voltage supply, so the voltage on the line rises to the higher of the two allowed levels if the line is not grounded. Since the 488 uses negative true logic, a line is given the value TRUE by turning the open-collector driver ON, or the value FALSE by turning the driver OFF. The phrases "active true" and "passive false" are used to describe this system; active true because the line must be actively connected to ground to impress a value of true on it, passive false because no action is needed (no connection is made) to make the value of the line false.

Each 488 device has one open-collector driver for each 488 line that it uses. More than one open-collector driver (that is, more than one 488 device) can be connected to each line. If all drivers are off the voltage on the line will be high, which means it has the value false. However, if one or more open-collector drivers are on, the line's voltage will be low, and it will have the value true. This is called a **"wire-or"** system because the logical value of the line is the logical OR of the logical values impressed on it by the several open-collector drivers connected to it. Thus each 488 device sends a true to the line by turning on its driver, or a false by turning the driver off. Note that if any device asserts a particular line true, that line will have the value true. However, if a device asserts a false (high) signal, it will be overridden by any device which asserts a true.

The eight data lines are named **DIO1** through **DIO8** (DIO stands for Data Input/Output). The least significant bit appears on DIO1, the most significant on DIO8. One point of possible confusion is that the data bits in an S-100 system are numbered 0 through 7, while the 488 data lines are numbered 1 through 8. Another is that S-100 systems assume positive true logic (high means TRUE, low means FALSE). Just remember that S-100 data bit 7 appears on DIO8, etc. and a 488 byte is the one's complement of an S-100 byte and everything should be all right.

The proper IEEE title for the three handshake lines is **"Data Byte Transfer Control"** lines. They are individually known as follows:

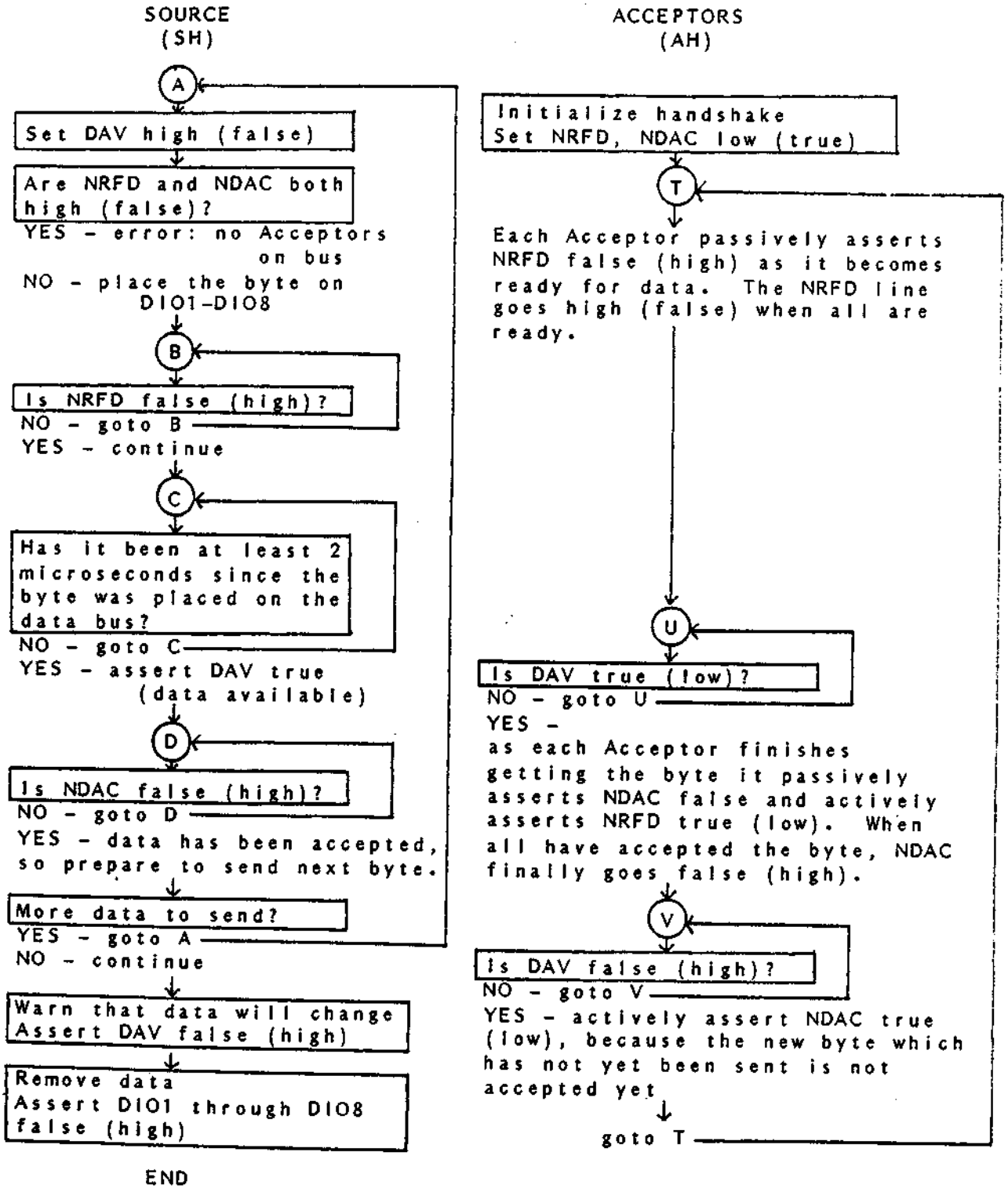
DAV (Data Valid) - when true the data on the eight data lines is valid.
NRFD (Not Ready For Data) - when true the 488 devices are not ready to accept data.
NDAC (Not Data Accepted) - when true the devices have not yet accepted the data.

The remaining five lines are known as the **"General Interface Management"** lines. They are as follows:

IFC (Interface Clear) - place all 488 devices in their default state.
ATN (Attention) - used to distinguish between a Controller and a Talker.
SRQ (Service Request) - indicates that a device needs attention.
REN (Remote Enable) - allows 488 devices to be programmed either by their local controls (front panel switches, etc.), or by information sent over the 488 bus.
EOI (End or Identify) - indicates the end of a string if ATN is false, otherwise it indicates a Parallel Poll is in progress.

- BYTE COMMUNICATION -

Byte communication requires that there be a device which is generating the byte to be communicated (the **"source"**) and one or more devices which receive the byte (the **"acceptors"**). The Source and Acceptors communicate by use of an interlocking handshake using the three Data Byte Transfer Control lines (DAV, NRFD and NDAC). The byte itself is sent on the eight data lines (DIO1 through DIO8). The handshake is schematized in the following flow chart.


SOURCE
(SH)
A
Set DAV high (false)
Are NRFD and NDAC both high (false)?
YES - error: no Acceptors on bus
NO - place the byte on DIO1-DIO8
B
Is NRFD false (high)?
NO - goto B
YES - continue
C
Has it been at least 2 microseconds since the byte was placed on the data bus?
NO - goto C
YES - assert DAV true (data available)
D
Is NDAC false (high)?
NO - goto D
YES - data has been accepted, so prepare to send next byte.
More data to send?
YES - goto A
NO - continue
Warn that data will change
Assert DAV false (high)
Remove data
Assert DIO1 through DIO8 false (high)
END
ACCEPTORS
(AH)
Initialize handshake
Set NRFD, NDAC low (true)
T
Each Acceptor passively asserts NRFD false (high) as it becomes ready for data. The NRFD line goes high (false) when all are ready.
U
Is DAV true (low)?
NO - goto U
YES -
as each Acceptor finishes getting the byte it passively asserts NDAC false and actively asserts NRFD true (low). When all have accepted the byte, NDAC finally goes false (high).
V
Is DAV false (high)?
NO - goto V
YES - actively assert NDAC true (low), because the new byte which has not yet been sent is not accepted yet
goto T

- A More Detailed Look at the 488 Inhabitants -

A **TALKER** is a device which sends data over the 488 interface to other devices. There are two major types and various subtypes. One major type is the Talk Only (ton), which may be used in a 488 system which has no Controller. This device always talks, and so it must be the only device which can talk. The other major type must be told when to talk ("**addressed to talk**"). A Controller is needed because it is the only kind of 488 device that is allowed to address Talkers and Listeners. All Talkers use the Source Handshake (SH) function to send a message over the 488 bus.

A **LISTENER** is a device which receives data over the 488 interface. As with the Talker, there are two major types: Listen Only (lon) and addressed Listener. A Listen Only device always listens to the 488 bus, while an addressed Listener listens only when the Controller tells it to. The Listen Only device can operate in a 488 system which does not have a Controller since it does not need to be told what to do and when to do it. All Listeners use the Acceptor Handshake (AH) function to receive messages on the 488 bus.

A **CONTROLLER** is a device which issues commands on the 488 bus. These include commands which are used to reset all devices on the bus Interface Clear (IFC), indicate which device is to Talk (when the Controller relinquishes the bus) and which devices are to Listen (i.e. it sends the Talk and Listen addresses of those devices over the bus), perform a Poll of 488 devices (Serial Poll and Parallel Poll), and a myriad of other special functions. The commands fall into two general classifications: **Uniline** and **Multiline**. Each uniline command uses only one line out of the five General Interface Management lines. Examples of uniline messages are Remote Enable (REN), Interface Clear (IFC) and Parallel Poll. Multiline messages use the eight data (DIO1-DIO8) lines to issue the command. Examples of multiline messages include performing a Serial Poll and commanding 488 devices to Talk or Listen. Multiline messages are sent using the Source Handshake (SH) function, just like a Talker. The way that a device determines whether it is hearing a Talker or the Controller is that the ATN (Attention) line is true (low) when the Controller is issuing a message, but false (high) when a Talker is saying something. The Controller is the device which controls the ATN line. Whenever ATN is true, all addressed Talkers shut up so that the Controller can say its piece. However, some Talk Only devices don't, and so they garble commands issued by the Controller. Generally speaking, a Talk Only device should be used only in a 488 system which has no Controller. Whenever the Controller passively asserts ATN false (lets it go high), the Talker (if any) begins to send its message.

- MULTILINE COMMANDS -

Telling a 488 device to Listen is one example of a multiline command. The Controller places the Listen address of the selected device on the data lines (DIO1 through DIO8) and then performs the Source Handshake (SH) function. In other words, it "speaks" the address while ATN is true (low). Whenever the Controller is active (that is, whenever ATN is true), all devices on the 488 bus interpret whatever is said (via the data lines and the Source Handshake function) as a command rather than data. ALL devices hear what is said by the Controller. They ALL execute the Acceptor Handshake function, without regard to whether they are normally a Talker, Listener or whatever.

Another example of a multiline command is the **Serial Poll.** The order of events is that the Controller sends out the Serial Poll Enable (**SPE**) command, which tells each device that when it is addressed as a Talker that it is to say either **SBN** (Status Byte - service Not requested) or **SBA** (Status Byte - service request Acknowledged). Those are the only two messages that are allowed. Then the Controller addresses each device as a Talker in turn and Listens to the response of each. To conclude a Serial Poll, the Controller sends the Serial Poll Disable (**SPD**) command so that any device later addressed as a Talker can speak data (instead of SBN or SBA). Finally, the Controller performs whatever service is needed, which is device dependent.

- UNILINE COMMANDS -

An example of a uniline command is **Parallel Poll.** Parallel Poll is both simpler and more complicated than Serial Poll. It is simpler because only one command is given (Identify **IDY**: the logical AND of **ATN** and **EOI**) and all devices respond at once. It is possibly more complicated in that it may be more difficult to sort out which device wants service. Whenever a 488 device receives the IDY message, it immediately places its Parallel Poll Response byte on the eight data lines. For systems of eight devices or less, it is common for each device to be assigned a unique bit which it asserts true when it needs service. For example, one device would have a Parallel Poll response byte in which bit 1 is true if it needs service, otherwise bit 1 is false, and bits 2 through 8 are always false. Another device would use bit 2 to indicate its need for service and all other bits would always be false in its response byte. A third device would use bit 3. When a Parallel Poll is performed, the response sensed by the Controller will be the logical OR of all the Parallel Poll Response bytes (due to the fact that the 488 bus is a wire-or system). If the response has bits 1 and 3 true, and all other bits false, it means that the first and third devices need service, while the second does not.

If the 488 system uses more than eight devices, some alternate scheme must be used. One would be to have only eight devices respond to a Parallel Poll, and use Serial Poll on the remaining devices. Another scheme would be to have several devices share the same Parallel Poll Response byte. If the response to a Parallel Poll shows that at least one of the devices that shares a common response needs service, a Serial Poll can be used to find which ones they are.

- OVERVIEW -

The P&T-488 has four read/write registers which appear as four input/output (I/O) ports to the S-100 host machine. The ports are addressed as four consecutive I/O ports with the first port address an integral multiple of 4 (0, 4, 8, 0C, ..., N*4, ..., FC). For ease of description these registers will be referred to as registers 0 through 3, even though what is called register 0 may be Port 0, 4, 8, ..., N*4, ..., FC.

The addresses used by the P&T-488 are set by means of a DIP switch on the upper left corner of the interface board. All boards are set at the factory for I/O ports 7C through 7F Hex, and all software supplied by Pickles & Trout assumes these addresses. The address used by both the board and the software can be changed by the user. The addresses used by the software and the board must be the same. To change the addresses assumed by the software, refer to the instructions given with the program.

To change the addresses used by the board, first note that the labels "A7" through "A2" appear to the left of the switch. Switches A2 through A7 are set according to the following table:

Address (Hex)	A7	A6	A5	A4	A3	A2
00-03	ON	ON	ON	ON	ON	ON
04-07	ON	ON	ON	ON	ON	OFF
08-0B	ON	ON	ON	ON	OFF	ON
0C-0F	ON	ON	ON	ON	OFF	OFF
10-13	ON	ON	ON	OFF	ON	ON
14-17	ON	ON	ON	OFF	ON	OFF
18-1B	ON	ON	ON	OFF	OFF	ON
1C-1F	ON	ON	ON	OFF	OFF	OFF
20-23	ON	ON	OFF	ON	ON	ON
24-27	ON	ON	OFF	ON	ON	OFF
28-2B	ON	ON	OFF	ON	OFF	ON
2C-2F	ON	ON	OFF	ON	OFF	OFF
30-33	ON	ON	OFF	OFF	ON	ON
34-37	ON	ON	OFF	OFF	ON	OFF
38-3B	ON	ON	OFF	OFF	OFF	ON
3C-3F	ON	ON	OFF	OFF	OFF	OFF
40-43	ON	OFF	ON	ON	ON	ON
44-47	ON	OFF	ON	ON	ON	OFF
48-4B	ON	OFF	ON	ON	OFF	ON
4C-4F	ON	OFF	ON	ON	OFF	OFF
50-53	ON	OFF	ON	OFF	ON	ON
54-57	ON	OFF	ON	OFF	ON	OFF
58-5B	ON	OFF	ON	OFF	OFF	ON
5C-5F	ON	OFF	ON	OFF	OFF	OFF
60-63	ON	OFF	OFF	ON	ON	ON
64-67	ON	OFF	OFF	ON	ON	OFF
68-6B	ON	OFF	OFF	ON	OFF	ON

Address (Hex)	A7	A6	A5	A4	A3	A2
6C-6F	ON	OFF	OFF	ON	OFF	OFF
7Ø-73	ON	OFF	OFF	OFF	ON	ON
74-77	ON	OFF	OFF	OFF	ON	OFF
78-7B	ON	OFF	OFF	OFF	OFF	ON
7C-7F	ON	OFF	OFF	OFF	OFF	OFF
8Ø-83	OFF	ON	ON	ON	ON	ON
84-87	OFF	ON	ON	ON	ON	OFF
88-8B	OFF	ON	ON	ON	OFF	ON
8C-8F	OFF	ON	ON	ON	OFF	OFF
9Ø-93	OFF	ON	ON	OFF	ON	ON
94-97	OFF	ON	ON	OFF	ON	OFF
98-9B	OFF	ON	ON	OFF	OFF	ON
9C-9F	OFF	ON	ON	OFF	OFF	OFF
AØ-A3	OFF	ON	OFF	ON	ON	ON
A4-A7	OFF	ON	OFF	ON	ON	OFF
A8-AB	OFF	ON	OFF	ON	OFF	ON
AC-AF	OFF	ON	OFF	ON	OFF	OFF
BØ-B3	OFF	ON	OFF	OFF	ON	ON
B4-B7	OFF	ON	OFF	OFF	ON	OFF
B8-BB	OFF	ON	OFF	OFF	OFF	ON
BC-BF	OFF	ON	OFF	OFF	OFF	OFF
CØ-C3	OFF	OFF	ON	ON	ON	ON
C4-C7	OFF	OFF	ON	ON	ON	OFF
C8-CB	OFF	OFF	ON	ON	OFF	ON
CC-CF	OFF	OFF	ON	ON	OFF	OFF
DØ-D3	OFF	OFF	ON	OFF	ON	ON
D4-D7	OFF	OFF	ON	OFF	ON	OFF
D8-DB	OFF	OFF	ON	OFF	OFF	ON
DC-DF	OFF	OFF	ON	OFF	OFF	OFF
EØ-E3	OFF	OFF	OFF	ON	ON	ON
E4-E7	OFF	OFF	OFF	ON	ON	OFF
E8-EB	OFF	OFF	OFF	ON	OFF	ON
EC-EF	OFF	OFF	OFF	ON	OFF	OFF
FØ-F3	OFF	OFF	OFF	OFF	ON	ON
F4-F7	OFF	OFF	OFF	OFF	ON	OFF
F8-FB	OFF	OFF	OFF	OFF	OFF	ON
FC-FF	OFF	OFF	OFF	OFF	OFF	OFF

For example, to address the P&T-488 interface board to use I/O ports 7C through 7F Hex, A7 must be ON and A2 through A6 OFF.

The P&T-488 allows direct access to the 8 signal lines of the IEEE 488-1978 (hereafter called 488) data bus (Register 2) and the 8 lines of the 488 Data Byte Transfer Control Bus and General Interface Management Bus (Register 1). In addition, a register is provided to allow a software settable response to a Parallel Poll (Register 3). Finally, a register is provided which indicates transitions occurring on the various 488 Control Bus and Management Bus lines (Register Ø). Additional features of the P&T-488 include software disable of interrupts from the P&T-488 (without having to disable all interrupts of the S-1ØØ system) and immediate response of the interface to Attention (ATN), Interface Clear (IFC) and Parallel Poll without intervention of the S-1ØØ system's CPU.

The data transfer rate is highly dependent on the software, CPU and system memory of the S-100 system, but with the supplied software, an 8080 running at 2.0 MHz and no memory wait states, the transfer rate is about 3 KBytes/sec. For applications requiring higher rates, the same S-100 system can get data rates of over 9 KBytes/sec in the Talk Only mode.

REGISTER FUNCTIONS

No.	FUNCTION
0	Interrupt Status (read only)
0	Interrupt Reset (write only)
1	Command Line Register (read and write)
2	Data Line Register (read and write)
3	Parallel Poll Response (write only)

REGISTER BIT MAP

No.	I/O	D7	D6	D5	D4	D3	D2	D1	D0
0	IN	DAV +-	NRFD +	NDAC +	XIFC -	XATN +-	SRQ -	REN +	POC -
0	OUT	DAV	NRFD	NDAC	XIFC	XATN	SRQ	TALK/ LISTN	DI/ EI
1	I/O	DAV	NRFD	NDAC	IFC	ATN	SRQ	REN	EOI
2	I/O	DIO8	DIO7	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1
3	OUT	DIO8	DIO7	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1

NOTES:

+ means the bit goes low on a LOW to HIGH transition
- means the bit goes low on a HIGH to LOW transition

DI means 488 interface interrupts are disabled
EI means 488 interface interrupts are enabled

The 488 data lines are numbered from 1 to 8, while the data lines on the S-100 system are numbered 0 to 7

X as in XATN, XIFC signifies that some device other than the P&T-488 has made the level on the line (ATN or IFC) active true (low).

- REGISTER 3 -

This register holds the Parallel Poll Response byte. Whatever has been output to Register 3 will appear on the 488 data lines in response to a Parallel Poll (ATN and EOI).

- REGISTER 2 -

This register is connected to the 488 data lines through bus transceivers. The state of the data lines can be sensed by reading Register 2, and the P&T-488 will assert on the data lines whatever was last written into Register 2. However, if either the XATN flag or XIFC flag in Register Ø is set, the output buffers to the 488 bus are disabled which precludes assertion of what was last written into Register 2. Remember that the 488 bus uses negative logic so that any bit that is low is asserted (or logically true). Also the 488 bus is a wire-or system, so if any piece of equipment is asserting a particular line true, that line will be a logical true. But if a device asserts a false (high) signal, it is overridden by any device that asserts a true. Hence the terminology of **active true** and **passive false**. Thus if the P&T-488 is being used as a Listener all bits of Register 2 should be written high (logic false) so that the data asserted by the Talker can be properly read.

- REGISTER 1 -

This register allows direct setting and sensing of the 488 Control and Management bus lines. If the XIFC flag is set in Register Ø, the interface will not assert any of the lines, regardless of what was last written into Register 1. Similarly, if XATN flag is set in Register Ø, the interface will not assert any line except Not Ready For Data (NRFD) and Service Request (SRQ). SRQ will be asserted active true (low) only if the SRQ bit (bit D2) of Register 1 was written low. NRFD will always be asserted active true (low). The reason that NRFD is asserted true is so that the System Controller will not send any commands until the S-1ØØ CPU is ready to accept them. Note that XATN has precedence over XIFC, so an externally applied IFC followed by an externally applied ATN will cause NRFD to be active true, SRQ to be true if the SRQ bit in Register 1 was written low, and all other 488 lines will be passive false.

- REGISTER Ø -

This is the Interrupt Status/Reset Register. Since the P&T-488 uses only one interrupt vector, one needs to be able to determine which condition caused the interrupt. Each bit of this register is associated with an interrupt-causing condition. By writing a low in the corresponding bits, one can individually reset the status bits associated with Data Valid (DAV), Not Ready For Data (NRFD), Not Data Accepted (NDAC), External Interface Clear (XIFC), External Attention (XATN) and Service Request (SRQ). If Bit 1 is set low status bit 7 will ignore any activity on the DAV line. This is useful when the interface is used as a Talker or Controller. If Bit 1 is set high, Bits 5 and 6 will ignore any activity on the NDAC and NRFD lines, which is useful when the interface is used as a Listener. If Bit Ø is set low, status Bits Ø (POC/RESET) and 1 (REN) will be cleared and the P&T-488 will be prevented from interrupting the S-1ØØ system (but the interrupt status bits will continue to respond to 488 Control and Management line activity). If Bit Ø

is set high the interface can interrupt the S-100 system.

If Bit 4 (IFC) of Register 1 is asserted there is no way of determining if an external Controller is also asserting IFC, so interrupt status bit 4 (XIFC) will ignore any activity due to an external Controller. A similar argument is true for ATN and XATN (Bit 3 of Registers 1 and 0). This is not a problem because the IEEE standard allows only the System Controller to assert IFC, and only the Controller-in-Charge may assert ATN. The standard further specifies that there may be no more than one System Controller and no more than one Controller-in-Charge.

P&T-488 Functional Test

The program 488TST81 performs seven different kinds of tests on the P&T-488 interface board and its 488 cable. The first group of four are done with no 488 device or test plug connected to the P&T-488. The last three are made with the special test plug connected to the P&T-488.

The program starts by printing a message to the operator to disconnect all 488 devices from the P&T-488. The operator signifies this has been done by pressing any key on the keyboard. After a key has been pressed the program begins its tests.

NOTE: Any time a Control C is pressed, the program is aborted and control is returned to the monitor (operating system).

The first test checks the data register (Register 2) by outputting a byte to the 488 data lines then reading the data lines to see if their state corresponds to the byte output to them. Each of the 256 possible bytes is tried in turn. If any errors occur, a message "**DATA ERROR – bits in error are ...**" with the bit names is printed. If there are no errors, no message is printed.

In a similar manner, the second test checks the command line register (Register 1). If there are any errors, the message "**COMMAND LINE ERROR – bits in error are ...**" is printed. Again, if there is no error, no message is printed.

The third test checks the Parallel Poll Response register (Register 3) by first making ATN and EOI true. Thus anything output to the Parallel Poll Response Register should appear on the 488 data lines. If the Command Line test failed with bits Ø and/or 3 in error, the results of this third test are meaningless. As with the first two tests, each of the 256 possible byte values is tried and any errors are reported: this time the error message is "**PARALLEL POLL ERROR – bits in error are ...**".

The fourth test checks the Interrupt Service Register (Register Ø). If the second test failed, this one will probably fail also. Errors are reported with the message "**INTERRUPT SERVICE REGISTER ERROR – bits in error are ...**".

After these four tests have been made, (they take less than a tenth of a second), the operator is told to attach the special test plug and then press any key on the keyboard to continue the tests. The plug connects the eight data lines to the eight 488 command lines, so that the 488 cable can be tested for continuity, shorts or incorrect wiring. It also allows testing the response of the P&T-488 board to ATN and IFC asserted true by an external Controller.

The fifth test checks the 488 cable and reports any bits in error. If either the first (data line) or second (command line) tests failed, the results of this test will be meaningless. If the first four tests were passed without error, but this one shows errors, it means either the cable and/or test plug is open, shorted, miswired or improperly plugged. If all bits are in error, the 488 cable is either not connected to the P&T-488 interface board or the special test plug is not plugged into the cable.

O.K. when no cable is connected
O.K. with TEKTRONIX CABLE
IOMAC CABLE CAUSES ERRORS.

The sixth test checks the response of the P&T-488 to an IFC (Interface Clear) presented by an external Controller. What is really done, of course, is to use the data port to assert a true on the IFC line through the special shorting plug, but the P&T-488 can't tell the difference between this and an external Controller making IFC true. The results are meaningful only if the first five tests passed with no errors.

The seventh test checks the response of the P&T-488 to an ATN (Attention) presented by an external Controller. The technique is the same as used in the sixth test. Again, the results are meaningful only if the first five tests were passed without any errors.

After the seventh test has been completed, the message NO ERRORS is printed if all tests were passed without error. Then the message **"P&T 488 functional test complete"** is printed and the program jumps back to the monitor.

WHAT TO DO IN CASE OF ERROR -

If any of the first four tests fail, check the following:

1. The P&T-488 interface board must be addressed to the same ports that the test routine tests. The base address (lowest address of the four) used by the P&T-488 must be in location 1Ø3 Hex for CP/M systems, 3ØØ3 Hex for North Star. The program is supplied with the base address set to 7C Hex.

2. All 488 devices must be disconnected from the P&T-488.

3. Make sure you are using the correct test routine. 488TST81 is to be used on ONLY Revision 81A boards (serial number 5ØØØ and up). 488TEST is to be used on ONLY boards with serial numbers under 5ØØØ.

If any of the first four tests fail, try disconnecting the 488 cable from the P&T-488 interface board. If they STILL fail, the P&T-488 is faulty and should be returned to Pickles & Trout for repair or replacement. Be sure to include a printout of the test results. If the first four tests are passed without error after the cable has been disconnected, the cable is defective (a short between lines or a short to ground).

If no error message is printed before the **"Attach test plug..."** message to the operator, the first four tests were passed without error. If the error message **"EXTERNAL ATN ERROR - bits in error are 2"** is displayed, it is likely that you are using the wrong test routine. 488TEST is to be used on only boards with serial numbers under 5ØØØ; 488TST81 is to be used only on boards with serial numbers over 4999. USE THE CORRECT TEST. If the error message "EXTERNAL INTERFACE CLEAR ERROR - ..." is printed with no error message preceding it, the P&T-488 is faulty. If the error message "EXTERNAL ATN ERROR - ..." is printed, and either there is no other error message or only the EXTERNAL INTERFACE CLEAR ERROR message, the P&T-488 is faulty and should be returned for repair or replacement.

RETURN POLICY -

The P&T-488 interface board, its 488 connecting cable and the special test plug are warranted to be free of defects in materials and workmanship for 90 days from the date of sale. If they should be found faulty within the warranty period, Pickles & Trout will

**** Introduction ****

The sequence that most people follow is

1. Unpack the P&T-488
2. Install it in an S-100 system
3. Test the P&T-488 to make sure it is operating properly
4. Write programs

The MSOFT portion of the P&T-488 manual will follow this sequence.

**** Unpacking the P&T-488 ****

The package contains the following items:

1. P&T-488 interface card
2. 18 inch cable
3. metric mounting hardware
4. P&T-488 test plug
5. floppy disk
6. manual
7. registration card

The 18 inch cable is designed to go from the P&T-488 card to the back panel of your S-100 computer. The 488 receptacle should be mounted with the metric mounting hardware provided: it is designed to mate with the jackscrews of standard 488 cables.

The floppy disk contains the MSOFT driver program, the P&T-488 functional self test program, some sample programs (so you can see real live examples of programs written for the MSOFT driver) and several utility programs.

The P&T-488 test plug is needed to perform the functional self test.

The registration card is very important! Please fill it out and mail it to us. It is our only means of getting your name and address so we can tell you of any bug fixes that we have come up with, inform you of new application programs and other things which will save you time and effort. Most of our orders come from purchasing departments, and they really are not interested in being notified about such things.

**** Installation ****

The P&T-488 interface card uses four contiguous I/O ports and is supplied configured to use ports 7C through 7F Hex (124 through 127 decimal). Be sure there is no port address conflict with other I/O boards in your S-100 system before installing the P&T-488. Refer to the chapter "Hardware Description" for instructions if it is necessary to change the I/O ports that the P&T-488 uses.

When you are satisfied that there is no I/O port address conflict between the P&T-488 interface and other devices in your S-100 system, turn off the power to the S-100 system and wait at least twenty seconds (to allow sufficient time for the S-100 power supply to discharge) before installing the P&T-488 card. Attach the cable to the back panel of the S-100 system using the metric hardware supplied with the cable (this hardware mates with the standard lockscrews used on 488 cables supplied by Hewlett-Packard, Beldon and others) and plug the cable onto the top connector of the P&T-488 interface card. Note that the plug and connector are keyed.

It will be necessary to modify 488TST81 if the I/O port addresses of the board have been changed from 7C through 7F Hex. The fourth byte in this program contains the lowest address of the four that is used by the P&T-488 interface card. If, for example, the card has been addressed to use ports 60 through 63 Hex you could change 488TST81 by following this procedure:

1. Load the program using the utility routine DDT (key "DDT 488TST81.COM"). Note that you are supposed to key what is in between the quote marks, but not the quote marks themselves. The mnemonic <CR> means to press the carriage return key. DO NOT type the four individual characters <, C, R and >.

2. Change the byte in location 103 Hex. (Key "S103<CR>". DDT will respond by displaying "103 7C" which is the address and the contents at that address. Then key the new base address: in this example it would be "60<CR>". DDT will then display the next memory location.)

3. Return to CP/M monitor. (Press and hold the Control key then press the letter C. Then release both keys.)

4. Put the modified file back on disk (key "SAVE 5 488TST60.COM<CR>"). Be sure not to use the file name 488TST81.COM.

As an example, assume that the port addresses used in 488TST81 are to be changed from 7C - 7F Hex to 60 - 63 Hex. Assume further that DDT is on disk drive A and 488TST81 is on drive B. Finally, assume that the new file is to be stored on drive B and its name is to be 488TST60 (the 60 is a reminder that this program is for the P&T-488 addressed to ports 60 - 63 Hex). The keys typed by the operator are underlined in the following dialog.

```
A>B:<CR>
B>A:DDT 488TST81.COM<CR>
DDT VERS 1.4
NEXT  PC
0600 0100
-S103<CR>
0103 7C 60<CR>
0104 00 †C
B>SAVE 5 488TST60.COM<CR>
B>
```

Note that the characters <CR> mean that the carriage return key is pressed not that the four characters <, C, R and > are typed. Also, the two character string †C means that the operator issued a Control C, not that the two keys † and C were typed.

****** Test the P&T-488 ******

Next the P&T-488 should be tested for proper operation. Run the program named 488TST81 and refer to the chapter "Functional Test" for instructions. After the test has been completed with no errors the 488 interface is ready for use.

****** Programs ******

MSOFT.REL	P&T-488 driver for compiler Basic
MSOFT.COM	P&T-488 driver for interpreter Basic
BCSAMPL.BAS	Compiler Basic program to exercise MSOFT
BISAMPL.BAS	Interpreter Basic program to exercise MSOFT
B488INIT.BAS	
BICLOCK.BAS	Interpreter Basic program to read an HP 59309 clock
CLOCK.MAC	Assembler program to read an HP 59309 clock
MTSAMPL.PAS	Pascal MT+ program to exercise MSOFT
MTCLOCK.PAS	Pascal MT+ program to read an HP 59309 clock
FSAMPL.FOR	Microsoft Fortran program to exercise MSOFT
FCLOCK.FOR	Microsoft Fortran program to read an HP 59309 clock
QCCLOCK.C	C program to read an HP 59309 clock

BUSMON.COM	IEEE-488 interactive bus monitor
488TODSK.COM	Put all 488 bus data into a disk file
DSKTO488.COM	Send contents of disk file as 488 data
HANDSHAK.ASM	Sample program for source and acceptor handshake
SAMPLHS.ASM	Sample program showing the use of HANDSHAK

Even though MSOFT is designed to work with Microsoft Basic, it can be used with some other languages as well. Programs written in assembler, C, Microsoft Fortran and Pascal MT+ are included to demonstrate how MSOFT can be used with these languages.

**** IEEE-488 Bus Monitor ****

A utility program named BUSMON is included on the software disk. This program is especially useful for experimenting and gaining familiarity with the 488 bus and the devices connected to it. The program is interactive and allows the user to send data as a Talker, commands as a Controller as well as send the various uniline messages (SRQ, REN, etc). The program is always a Listener and reports immediately any data, commands or uniline messages which appear on the 488 bus. BUSMON and the other utility programs are described in detail in the chapter "P&T-488 Auxiliary Programs for CP/M". This chapter appears at the end of the manual.

**** Sample Basic Programs ****

The Basic programs BISAMPL and BCSAMPL are also useful for dinking around and gaining familiarity with the 488 bus, the P&T-488 interface and whatever instruments are connected to the 488 bus. BISAMPL is a version written for the Microsoft Basic interpreter (MBASIC) and BCSAMPL is the same program written for Microsoft's Basic compiler (BASCOM). BUSMON has more capability and is more useful for actually debugging 488 bus operation, while BISAMPL and BCSAMPL are written in Basic and can serve as examples of how to write programs which use MSOFT.

The general form of the command line to load and run Basic programs which use the P&T-488 and interpreter Basic is the following:

x:MSOFT y:filenam1 z:filenam2<CR>

where x is the drive on which the program MSOFT is mounted
y is the drive on which the file **filenam1** is mounted
z is the drive on which file **filenam2** is mounted
filenam1 is the name of the Basic interpreter/run time package
filenam2 is the name of the Basic program itself.

For example, if MSOFT is on drive A, MBASIC is on drive C and BISAMPL is on drive B, the command line would be

A:MSOFT C:MBASIC B:BISAMPL<CR>

As is normal with CP/M, you do not need to specify the drive name if it is the current default drive.

**** MSOFT: The P&T-488 Driver Program ****

There are two versions of MSOFT on your disk: MSOFT.COM and MSOFT.REL. MSOFT.COM is the version to be used with the interpreter Basic, and MSOFT.REL is to be used with compiling Basic.

The program MSOFT is an interface between Microsoft Basic Rev 5.00 (and later) and the

IEEE-488 bus. You can use MSOFT to perform the following functions:

488 Bus Control	Remote Enable	Parallel Poll
Talk	Local	Serial Poll
Listen	488 Interface Clear (IFC)	

MSOFT is designed to allow you to easily access the IEEE-488 bus from either the compiler or interpreter version of Microsoft Basic. It uses a calling convention which is easy to understand and use, and which also provides the 488 functions commonly needed in a laboratory or automated test facility.

A typical application program consists of two parts: a Basic program and MSOFT (a machine language program). Thirteen communication functions are available to allow the Basic program to be a Controller, Talker or Listener on the 488 bus, as well as perform other 488 operations. These functions use eleven variables to control communication between the Basic program and MSOFT. These variables may assume any legal Basic variable name. MSOFT functions are executed by using Basic CALL statements and passing the appropriate parameters.

**** How It Works ****

The key to the operation of MSOFT is the CALL statement. CALL statements are of the form

CALL <variable name> (<parameter 1>, <parameter 2>, ... ,<parameter N>)

where <variable name> is the name of the variable which contains the address of the machine language routine you want to call, and <parameter1>, <parameter2>, etc., are the parameters you want to pass to the subroutine. You may pass any number of parameters, but the number and type of parameters passed must match the number and type of parameters expected by the machine language subroutine. Note that a passed parameter cannot be a constant or a string literal (e.g. 27.5 or "hello there").

When Basic passes a variable via the CALL statement it doesn't actually pass the variable itself, but only a pointer to the variable. If the variable is an integer the pointer points to the number itself. Integers, which is the only type of numeric variable that MSOFT uses, are stored as two byte-two's complement numbers, low order byte first. If the variable is a string the pointer points to that string's string descriptor. String descriptors consist of two parts: the string length (one byte), and the address in memory where the string is stored (two bytes, low order byte first).

When one of the MSOFT setup routines is called Basic passes the appropriate pointers to MSOFT. MSOFT then transfers these pointers to a table so it can remember which variable names you are using for the various communication variables. In this way MSOFT can automatically read (or write to) the variables used for Basic-MSOFT communication. If you need to change some communication parameter all you have to do is assign the parameter a different value and MSOFT will automatically note the change.

Example: Suppose you want to turn the input echo function on and off. If you named the variables for input and output echo ECHOIN% and ECHOOUT% respectively, you would say

```
100 CALL ECHO(ECHOIN%,ECHOOUT%)
```

to tell MSOFT the names of the input and output echo variables. Since Basic always initializes variables to zero, both the input and output echo functions are initially off. When you want to turn on the input echo you need only make ECHOIN% non-zero, as is shown by the following program line:

```
135 ECHOIN%=1
```

In this example we have shown the two basic units needed to communicate with MSOFT. One is the "Communication Function" (in this case ECHO) and the other is the "Communication Variable" (ECHOIN% and ECHOOUT%).

**** Communication Functions ****

There are thirteen communication functions, four setup functions and one configuration function available to the user in the MSOFT program. The communication functions — as their name implies — control data transfer and housekeeping on the 488 bus. The setup functions are used to inform MSOFT what variables to use to communicate with a Basic program. These functions are invoked by using a Basic call to the setup function SETUP%. You may use different names for the communication functions in a program which is to be used with the interpreter version of Basic, but you must use the names shown below if you use the compiler version. The names shown here are the ones used in the sample programs and in the file INIT.BAS, which has all the code required to set up communication between MSOFT and your Basic program. The configuration function is used to tell MSOFT what I/O ports the P&T-488 board is using.

The parameters which are used by the communication and setup functions fall into two general categories: output variables and input variables. Output variables are values you send to MSOFT. Input variables are values that MSOFT sends to you. Each of these categories is broken down into two subcategories according to the type of variable used: integers and strings. The communication functions use only strings while the setup functions use only integers.

The communication functions are:

1. CNTL% (<output string>)

   Example: 100 CALL CNTL% (A$)

   Become the 488 Controller and send the output string as a command string over the 488 bus. The error code is updated by this function.

2. CNTLC% (<output string>)

   Works like CNTL%, but the error code is set equal to zero (cleared) before transmitting the command string.

3. TALK% (<output string>)

   Example: 100 CALL TALK% (A$)

   Become a Talker and transmit the output string over the 488 bus. All 488 data lines are left passive FALSE after the last byte has been sent. NOTE: see EOT switch and EOS value. The error code is updated by this function.

4. TALKC% (<output string>)

   Works like TALK%, but the error code is cleared before the output string is transmitted.

5. LSTN% (<input string>)

   Example: 100 CALL LSTN% (A$)

   Become a Listener and receive an input string over the 488 bus. The NRFD line is left true after receiving the last byte. NOTE: see EOT switch and EOS value. The error code is updated by this function.

6. LSTNC% (<input string>)

Works like LSTN%, but the error code is cleared before receiving the input string.

7. SPOLL% (<output string>, <input string>)
Example: 100 CALL SPOLL% (A$,B$)

Perform a Serial Poll by sending the Serial Poll Enable (SPE) message as a Controller, then sending UNTALK followed by the first Talk Address in the output string. SPOLL% then gets a single byte from the newly addressed Talker and checks it to see if that Talker is requesting service. If that Talker was not requesting service, SPOLL% sends UNTALK followed by the next Talk address in the output string and gets that Talker's response byte. It continues doing so until it either finds a device requesting service, encounters an invalid Talk Address, has tried all addresses in the output string, or encounters a bus error. If it finds the device requesting service it puts the poll response byte in POLL% and the device's Talk Address in the input string, sends UNTALK followed by Serial Poll Disable (SPD) as a Controller, then returns to Basic. If it encounters an invalid Talk Address or tries all addresses but does not find the device requesting service, it makes the input string a null string, sends UNTALK followed by Serial Poll Disable (SPD) as a Controller, then returns to Basic. If it encounters a bus error (timeout, IFC, etc.), it puts the error code in the error code byte, makes the input string a null string and returns to Basic. NOTE THAT IT DOES NOT SEND UNTALK OR SERIAL POLL DISABLE!! It cannot because of the bus error, and the other devices on the bus may well be left in the Serial Poll mode instead of the Data mode of operation. It is up to your program to take whatever action is appropriate in case of error. (One possibility is to send an IFC, which resets all 488 devices to their initial state. However, in some cases this may not be appropriate.)

Note that the output string should contain only the Talk addresses of the devices to be polled. If the Talk Address of some device which is not connected to the bus is in the output string, SPOLL% will address it to talk and wait for its response. None is forthcoming, since the device is not connected! The result will be either a timeout error, or, if the timeout function has been disabled (by setting the time value to 255), the 488 bus and your S–100 system will lock up. The only recovery to such a lock–up is for you to reboot your S–100 system.

Note also that the poll response variable is updated only when the device requesting service is found. If no such device is found POLL% contains whatever garbage it had when SPOLL% was called. You can tell whether the contents of POLL% are meaningful by looking at the input string: it is a null string (has a length of zero) if the device requesting service was not found. Otherwise it is a non–null string which is the Talk Address of the device requesting service.

8. PPOLL% Example: 100 CALL PPOLL%

Performs a Parallel Poll of the 488 devices (by making the 488 ATN and EOI lines true). The response is placed in the poll response variable. Note that no arguments are used with the PPOLL% call. This function does not affect the error code.

9. DREN% Example: 100 CALL DREN%

Make the REN (Remote Enable) line of the 488 bus false, which places all devices in their LOCAL mode. This function does not affect the error code.

10. REN% Example: 100 CALL REN%

Make the REN line of the 488 bus true. Once the REN line goes true any device addressed as a Listener by the Controller will enter the remote mode. This function does not affect the error code.

11. STATUS% Example: 100 CALL STATUS%

Calling this function updates the bus status variable. STATUS% allows the user to determine the bus status without becoming a Controller, Talker or Listener. It is primarily used to determine if some special condition is occurring on the 488 bus (another Controller issuing an IFC, etc). The way it works is that it checks for XATN, XIFC, POC and SRQ. It then sets the appropriate bits of the error code and then copies the five most significant bits of the error code into the bus status variable. The three least significant bits of the bus status variable are set to zero. Notice that since STATUS% does not reset the error code before checking the 488 bus, the error code and the bus status variable may show a condition which occurred before STATUS% was called. The error code is updated by this function.

12. IFC% Example: 100 CALL IFC%

Initialize the bus. This function resets the P&T 488 and then issues an IFC (Interface Clear), which puts all 488 devices in their default state. It terminates with the NRFD line true, which prevents any communication from taking place on the 488 bus until the MSOFT system is ready to participate. This function does not affect the error code.

13. BRSET% Example: 100 CALL BRSET%

Resets the P&T 488. Unlike the IFC% command, it does not send an IFC nor does it make NRFD true. Thus if it is desired to allow communication to take place on the 488 bus without the participation of the Basic program, one can use the BRSET% call. This function does not affect the error code.

****** Setup Functions ******

Since you are allowed to choose the names you want for the variables used to communicate with MSOFT, you must tell it the names of the variables. There are four setup functions that are used for this purpose:

1. SETUP% (<CNTL%>, <CNTLC%>, <TALK%>, <TALKC%>, <LSTN%>, <LSTNC%>, <SPOLL%>, <PPOLL%>, <DREN%>, <REN%>, <STATUS%>, <IFC%>, <BRSET%>, <IOSET%>, <PROTCL%>, <ECHO%>, <IOPORT%>)

This function is needed only for a program which is to be run with the interpreter version of Basic. You do not need to use this function if the program is to be run with the compiling version of Basic because the compiler already "knows" the names of the communication functions. In fact, you cannot use this function since the compiler allows a maximum of ten parameters to be passed through a CALL.

This function sets up the variable names that MSOFT is to use for all the 488 bus functions and the following three setup functions. Note that the value of SETUP% must be calculated. The calculation can be performed by the following three lines of code:

```
100 TEMP = 256*PEEK(7)+PEEK(6)+9
110 IF TEMP > 32767 THEN TEMP = TEMP-65536!
120 SETUP% = CINT(TEMP)
```

Line 100 calculates the address of the setup function in MSOFT. Line 110 ensures that the value of TEMP is in the range of -32768 to 32767 (which is the range of an integer). Line 120 sets SETUP% to the integer value of TEMP.

NOTE: You must call SETUP% BEFORE you make use of any other MSOFT function if you are using interpreter Basic. SETUP% is the function that establishes all the "hooks" needed by MSOFT to communicate with your basic program.

2. IOSET% (<error code>, <timeout value>, <poll result>, <bus status>)

This function sets up the variable names that MSOFT is to use for the error code, timeout value, poll result, and the bus status. It sets a default timeout value of 254 each time it is called. This function must be called before using any of the function calls.

Example: 100 CALL IOSET% (ERCODE%, TIME%, POLL%, BUS%)

3. PROTCL% (<EOT switch>, <EOS value>, <string length>)

This function sets up the variables for the data transfer protocol. It sets the default string length to 254 each time it is called. This function must be called before using any of the function calls.

Example: 100 CALL PROTCL% (EOT%, EOS%, LENGTH%)

4. ECHO% (<input echo flag>, <output echo flag>)

This function sets up the variables for the input and output echo flags. This function call is optional. If it's not called before using any of the communication function calls, the default value is no input or output echo.

Example: 100 CALL ECHO% (ECHOIN%, ECHOOUT%)

**** NOTE ****

Basic does not have any mechanism to check that the correct number and type of variables are passed by a CALL function, so MSOFT cannot determine whether the arguments are valid. Thus it is extremely important that when you call one of the MSOFT functions that you use the right number of arguments, and the right type of arguments. Never, NEVER do a call with the wrong number of arguments, or with arguments of the wrong type. If you do, your program will most likely fail and give unpredictable results. The best thing to do is to be extra careful when typing in statements involving MSOFT function calls.

****** Configuration Function ******

The P&T-488 board is shipped from the factory set up to use S-100 I/O ports 7C through 7F Hex (124 through 127 decimal). If your S-100 system already uses these ports for some other function, the P&T-488 must be re-addressed to some other set of ports. The section of the manual titled "Hardware Description" tells you how to change the address of the P&T-488 board. You will also have to tell MSOFT what the new address is. MSOFT assumes that the P&T-488 board uses addresses 7C through 7F. By calling IOPORT you can tell MSOFT the lowest address used by the P&T-488 board.

For example, assume that you change the P&T-488 board so it uses ports 38 through 3B Hex. The following program line will tell MSOFT this new address:

```
100  PORT%=56 : CALL IOPORT%(PORT%)
```

PORT% was set to 56 decimal, which is the equivalent of 38 Hex (the lowest address used by the P&T-488 in this example). Note that PORT% must be set before calling IOPORT! IOPORT is not like the communication functions (IOSET, PROTCL, etc) because it passes to MSOFT the value of the parameter, while the communication functions pass to MSOFT the names of the parameters. You can change the timeout value at any time without having to call IOSET again, but you cannot change the port numbers without calling IOPORT again.

This was done on purpose. Your program should call IOPORT no more than once since you will not be changing the port numbers used by the P&T-488 while the program is running. If IOPORT told MSOFT the name of the parameter, and you used that parameter again later on in the program for something else, MSOFT would then try to communicate with the P&T-488 using incorrect port numbers.

The rules for the use of IOPORT are simple but important. You need to use it only if you have re-addressed the P&T-488 to some address other than 7C through 7F Hex. If the P&T-488 has been readdressed, you must use it after you call SETUP but before you use any communication function (CNTL, TALK, etc).

**** Communication Variables ****

The variables used fall into two categories: output variables and input variables. Output variables are values you send to MSOFT. Input variables are values that MSOFT sends to you. The purpose and type of each of these variables is listed below. In each case a variable name is also shown. You do not have to use this variable name, but it is the one used in the sample programs and in the file INIT.BAS, which has all the code required to set up communication between MSOFT and your Basic program.

1. ERROR CODE ERCODE% (integer, input variable)
   This variable indicates what errors (if any) occurred while using the 488 bus functions. It is sometimes called the RETURN CODE. The variable is a sixteen bit integer, while the error code is only eight bits. The error code is contained in the lower eight bits of the error code variable. Each bit is associated with a particular error condition. If the bit has the value "1" the corresponding error has occurred.

   0000 0000 Normal return - the function has been successfully completed. (Notice that no bit is set to "1").

   1... .... The S-100 RESET line is/has been true.

   .1.. .... The IFC line on the 488 bus has been true. Re-initialize the P&T 488.

   ..1. .... The ATN line on the 488 bus is/has been true. An external 488 Controller is trying to issue a command. (MSOFT will not work with 488 systems which have another Controller on the 488 bus.)

   ...1 .... Bus timeout error. No handshake has taken place in the allotted amount of time.

   .... 1... The SRQ line on the 488 bus is true. Some 488 device wants service. Refer to the manufacturer's manual to determine what action is necessary.

.... .1.. Serial Poll address error. An invalid Talk address is in the string of devices to be polled.

.... ..1. No Acceptors on the 488 bus. If this error occurs while performing a Control function, it means that there are no 488 devices on the bus which are capable of being addressed or programmed by the Controller. If this error occurs during a Talk function, it means that there are no 488 devices on the bus which are listening.

....1 Either IOSET% or PROTCL% was not called before trying to use one of the MSOFT communication functions. This error code will not tell you if the wrong number of arguments was passed to either IOSET% or PROTCL%, it will only tell you if one of them wasn't called prior to calling an MSOFT communication function.

Example: If the value of the error variable is found to be 192 (1100 0000 binary), it means that BOTH the 488 IFC line AND the S-100 RESET are or have been true.

Functions CNTLC%, TALKC% and LSTNC% reset the error code before they begin 488 bus communication. They then set the appropriate bits (if any) before returning to the Basic program. Functions CNTL%, TALK%, LSTN%, SPOLL% and STATUS% do not reset the error code before they begin 488 bus communication. They do set the appropriate bits (if any) before returning to Basic. Thus the error code may show errors which have occurred before these functions were called.

The eight functions CNTL%, CNTLC%, TALK%, TALKC%, LSTN%, LSTNC%, SPOLL% and STATUS% are the only functions which affect the error code.

2. TIMEOUT VALUE TIME% (integer, output variable)
This variable sets the amount of time within which a 488 handshake cycle must occur or else a bus timeout error will occur. As with the error code, MSOFT only uses the lower eight bits of this variable: the actual value used is the timeout value modulo 256. If it is set to 255 Decimal, no timeout check is made; that is, even if a handshake cycle is never completed, a timeout error is not generated. For a value of 0 through 254 Decimal, the value is used to indicate the amount of time that the handshake may take before a timeout error is generated. The amount of time that the timing loop takes varies with the processor (8080 or Z-80), system clock rate, etc. On an 8080 system running at 2 MHz a value of 200 corresponds roughly to 5 seconds or a value of 4 corresponds to about 100 milliseconds.

NOTE: The TIMEOUT value is set to 254 each time you CALL IOSET%.

3. POLL RESPONSE POLL% (integer, input variable)
The lower eight bits of the poll response variable contain the response to the most recent Serial or Parallel Poll.

4. BUS STATUS BUS% (integer, input variable)
The bus status tells the user the current bus state. Note: to save time, the bus status is not automatically updated as the bus state changes. The bus status function must be called each time the bus status is desired. The coding used is exactly the same as that used for the error code except that only the five most significant bits are used. The three least significant bits are always set to zero.

0000 0000 Normal return - the function has been successfully completed. (Notice that no bit is set to "1").

1... .... The S-100 RESET line is/has been true.

.1.. .... The IFC line on the 488 bus has been true. Re-initialize the P&T 488.

..1. .... The ATN line on the 488 bus is/has been true. An external 488 Controller is trying to issue a command. (MSOFT will not work with 488 systems which have another Controller on the 488 bus.)

...1 .... Bus timeout error. No handshake took place in the allotted amount of time during the previous TALK%, TALKC%, LSTN%, LSTNC%, CNTL%, CNTLC% or SPOLL% function.

.... 1... The SRQ line on the 488 bus is true. Some 488 device wants service. Refer to the manufacturer's manual to determine what action is necessary.

5. EOT SWITCH EOT% (integer, output variable)

This variable tells MSOFT how to recognize the end of a data transmission (if it's a Listener), or what to send at the end of its data transmission (if it's a Talker). There are three ways to specify the end of a data transmission: 1) The data transmission is assumed to be finished after a certain number of characters. 2) The data transmission is assumed to end with an END message. 3) The data transmission is assumed to end with a special end-of-string (EOS) character. The EOT switch can be greater than zero, zero, or less than zero.

LISTEN MODE:

EOT > 0 Terminate string collection upon receipt of an EOS character, END or if the LENGTH is matched.

EOT = 0 Terminate string collection upon receipt of END or if the LENGTH is matched.

EOT < 0 Terminate string collection upon receipt of END or if the LENGTH is matched. (Same as EOT = 0.)

TALK MODE:

EOT > 0 Append the EOS character to the end of the string.

EOT = 0 Send string as-is.

EOT < 0 Send the END message with the last byte of the string.

6. EOS VALUE EOS% (integer, output variable)

If the value of the EOT switch is greater than zero MSOFT looks for (or sends) this value as the end of a data transmission. Since there are only eight bits of data on the IEEE-488 bus, MSOFT only uses the lower eight bits of the EOS value.

7. STRING LENGTH LENGTH% (integer, output variable)

This is used only in the LISTEN mode (that is, when you CALL LSTN% or CALL LSTNC%). MSOFT uses this variable to determine the length of incoming messages. For instance, if the string length was set to 25 Decimal, then MSOFT would assume a data transmission was over after receiving 25 characters.

NOTE: The string length is set to a default value of 254 each time you CALL PROTCL%.

8. INPUT ECHO FLAG ECHOIN% (integer, output variable)
If the input echo flag is non-zero, then characters received by MSOFT are echoed to the console, otherwise they're not. The default value is zero.

9. OUTPUT ECHO FLAG ECHOOUT% (integer, output variable)
If the output echo flag is non-zero, then characters sent by MSOFT are echoed to the console, otherwise they're not. The default value is zero.

10. OUTPUT STRING (string, output variable)
The output string is the string of characters or commands that you wish to send over the 488 bus. Remember, you don't need an EOS character in your output string. MSOFT will automatically generate an EOS character, an END message, or nothing at all, depending on how the EOT switch is set.

11. INPUT STRING (string, input variable)
The input string is the string most recently received by MSOFT. If the EOT switch is positive (EOS selected), then MSOFT will automatically remove the EOS character from the end of the string.

Note: the input string variable and the output string variable may have the same name.

As you may have noticed, the input and output string variables are not passed to MSOFT through the setup functions. There is a method to the madness, however. While the values of MSOFT numeric variables might change frequently, it shouldn't be necessary to change the names of these variables very often, if at all. For instance, if you called the timeout variable TIME%, you may change its value many times, but there is little need to change its name to something else. However, when using the MSOFT string variables, there are many occasions where it would be nice to change the names of the variables used. You could just change the contents of a string variable (by using an assignment statement like '100 A$ = B$'), but string assignments take a comparitively long time, and it's faster just to pass the desired string variable (B$ in this case).

For example, if you have a standard programming string for each instrument on the bus, it is simpler to say

```
100 CALL TALK%(HP3455$)
110 CALL TALK%(HP9876$)
```

than it is to say

```
100 A$=HP3455$
110 CALL TALK%(A$)
120 A$=HP9876$
130 CALL TALK%(A$)
```

****** Quirks, Oddities and Strange Behavior ******

The following characteristics of MSOFT may give rise to unexpected results. The user should be aware of these characteristics so that they may be used to aid, rather than hinder, program development.

1. The CONTROLLER functions CNTL and CNTLC return to Basic with ATN true. The reason is that in some cases the user may want to send out several different strings and not have ATN go false in between them. For instance, the user can write

```
310 CALL CNTL%(A$)
320 CALL CNTL%(B$)
330 CALL CNTL%(C$)
```

and have all three strings be sent as a Controller without ATN going false in between them. A case in which this might be desirable is shown in the following program fragment:

```
310 DVMTLK$="T"
320 LCRTLK$="W"
330 L9876A$="3"
340 L2631$="6"
350 UNT$=CHR$(95)
360 UNL$=CHR$(63)
370 PRINT "CODE    INSTRUMENT"
380 PRINT "  1     HP 3455A DVM
390 PRINT "  2     HP 4275  LCR METER
400 INPUT "What instrument do you want to TALK (1 or 2)";TLKNO%
410 PRINT "CODE    INSTRUMENT"
420 PRINT "  1     HP 9876A PRINTER
430 PRINT "  2     HP 2631 PRINTER
440 INPUT "What instrument do you want to LISTEN (1 or 2)";LSNO%
450 CALL CNTLC%(UNT$)
460 CALL CNTL%(UNL$)
470 IF TLKNO%=1 THEN CALL CNTL%(DVMTLK$) ELSE CALL CNTL%(LCRTLK$)
480 IF LSNO%=1 THEN CALL CNTL%(L9876A$) ELSE CALL CNTL%(L2631$)
```

If CNTL and CNTLC made ATN false before returning to Basic the selected Talker would try to send data over the bus as soon as line 470 is executed. Since the Listener had not been designated yet the Talker would abort with a "No Listener" error. ATN will be made false when LSTN%, LSTNC%, IFC% or BRSET% is called.

2. A related topic involves the PARALLEL POLL function PPOLL. It also leaves ATN true when it returns to Basic. The idea is that after a parallel poll the user usually wants to become a Controller and issue some commands which are based on the results of the parallel poll. Thus the way it is set up now ATN remains true between the time of the parallel poll and the use of the Controller functions. If ATN were made false by the parallel poll function before it returns to the Basic program, there would be a period between the poll and the beginning of the Controller function during which 488 data communication can proceed.

3. The STATUS function updates the error code and then copies the appropriate bits into the bus status variable.

4. The way the error code is presently set up is that the bus communication functions (TALK, TALKC, LSTN, LSTNC, CNTL, CNTLC, SPOLL and STATUS) can <u>set</u> error bits, but only TALKC, LSTNC, CNTLC, BRSET and the user can <u>clear</u> error bits. The reasoning is that you may want to do a series of bus functions and check for error only after they are all done (which considerably speeds up bus communication). If the error code showed only what (if any) errors occurred during the <u>most recent</u> bus communication function, you would have to keep and update your own cumulative error flag, which would completely negate any speed improvement.

5. IOSET always sets a default timeout of 254 and PROTCL always sets a default string length of 254. They do this so that the system will work even if the user forgets to initialize TIME% and LENGTH%. (Remember that Basic always initializes integer variables to 0, so if IOSET and PROTCL did <u>not</u> set default values and the user forgot to set the timeout or string length he would almost always get timeout errors, and never get a listen string because the string length indicated that zero characters are to be gathered from the bus.)

6. One problem that often rears its ugly head has to do with how 488 devices terminate a

message. Some use the END message (EOI true on the last byte), some use a fixed length message and some use a single End-Of-String (EOS) character. All of these techniques are easily handled by MSOFT. However, there are some devices which use more than one character to indicate the end of a message: the usual multiple character end of string message is a carriage return followed by a line feed. The "correct" way to set up MSOFT in this case is to tell it to look for an EOS character, and tell it that the EOS character is a line feed. The problem is that the string you get back from MSOFT contains a carriage return as the last character. At times this can be a real bother. One way of dealing with the problem is to copy all but the last character of the string into another string with the statement

```
100 NEW$=LEFT$(OLD$,LEN(OLD$)-1)
```

7. MSOFT does not automatically start up in the RESET state. You must do a CALL BRSET in your application programs before you try to do any other bus function.

8. SPOLL will leave the bus in the Serial Poll mode instead of the Data mode if it encounters a bus error (handshake timeout, IFC, etc.). Thus if SPOLL is interrupted by a bus error you must restore the bus to data mode. This can be done by issuing an Interface Clear (IFC), or by clearing the bus error then sending out Serial Poll Disable (SPD) as a controller.

***** Gotchyas *****

Gotchyas (sometimes called "features" by advertising types) are characteristics of a product which are almost certain to bite the user in a most tender, if not vital, spot. Gotchyas are usually the result of either a lack of care in the design of the product, or are due to limitations over which the manufacturer has no control. MSOFT's known gotchyas fall into the latter category. We have done what we can to limit their number and effect, but the ones we know about are either unavoidable, or the result of avoiding them is to create even more of them. If you find more gotchyas, please let us know so that we can warn others of their existence and possibly get rid of them.

Gotchya Number 1

Basic does not have any mechanism to check that the correct number and type of variables are passed by a CALL function, so MSOFT cannot determine whether the arguments are valid. Thus it is extremely important that when you call one of the MSOFT functions that you use the right number of arguments, and the right type of arguments. Never, NEVER do a call with the wrong number of arguments, or with arguments of the wrong type. If you do, your program will most likely fail and give unpredictable results. The best thing to do is to be extra careful when typing in statements involving MSOFT function calls.

Gotchya Number 2

MSOFT does not perform an automatic reset when it starts up. You must do a CALL BRSET% or a CALL IFC% before you perform any other 488 bus function which looks at the error code (TALK, TALKC, LSTN, LSTNC, CNTL, CNTLC or SPOLL). You need to do this only once (it is a bus initialization step). If you neglect to do a CALL BRSET% or a CALL IFC% before the first time you call TALK, TALKC, etc, you will most likely get an S-100 RESET error, as well as several others.

Gotchya Number 3 (Occurs only with LSTN and LSTNC)

The way that MSOFT passes a string back to Basic is by dinking with the string address in the string descriptor area. MSOFT has its own 256 byte buffer to hold any string heard on the 488 bus, and it changes Basic's descriptor area to point to this buffer. Everything is OK until you go to get the next string by LSTN or LSTNC. If that string has a different name, what you wind up with is two different string names both pointing to the MSOFT string buffer, so the contents of both strings will be the same.

For example, if you have a program that looks like this

```
100 CALL LSTN%(A$)          :!**** THIS CODE WILL NOT WORK ****
110 CALL LSTN%(B$)          :!**** THIS CODE WILL NOT WORK ****
```

both A$ and B$ will point to the MSOFT string buffer and will both contain the string heard with the second LSTN command. The string heard by the first LSTN command will be lost. If you want to get two or more strings from the bus as a Listener without losing the contents of the earlier strings, you can write your program like this:

```
100 CALL LSTN%(DUMMY$)      :!**** THIS CODE WILL WORK ****
105 A$=DUMMY$               :!**** THIS CODE WILL WORK ****
110 CALL LSTN%(DUMMY$)      :!**** THIS CODE WILL WORK ****
115 B$=DUMMY$               :!**** THIS CODE WILL WORK ****
```

Statements 105 and 115 cause Basic to copy the contents of DUMMY$ (which happens to be the string buffer in MSOFT) into strings A$ and B$, respectively. Since A$ is a copy of what was heard on the bus statement 110 will not destroy it.

If you do not need to preserve the previous message, and, in fact, use the same string variable over and over, you do not need to worry about this problem. For instance, if you are waiting for a 488 device to send the string "QUIT" and you want to ignore all others, the following program segment will work just fine.

```
100 CALL LSTN%(A$)          :!**** THIS CODE WILL WORK ****
110 IF A$<>"QUIT" THEN 100  :!**** THIS CODE WILL WORK ****
```

The time you have to really watch for this problem is when you want to remember previous messages. If you are trying to get a set of readings from an instrument and you want to keep them in an array, the following code will not work.

```
100 FOR I%=0 TO 35
110 CALL LSTN%(A$(I%))      :!**** THIS CODE WILL NOT WORK ****
120 NEXT I%
```

What will happen is that A$(0), A$(1), ... will all point to the buffer in MSOFT, and it will hold only the last reading. The following code will work.

```
100 FOR I%=0 TO 35
110 CALL LSTN%(DUMMY$)      :!**** THIS CODE WILL WORK ****
120 A$(I)=DUMMY$            :!**** THIS CODE WILL WORK ****
130 NEXT I%
```

Gotchya Number 4

The Serial Poll function SPOLL can leave the bus in a state where the Talker will send only its serial poll response byte instead of data. This occurs only if a bus error (timeout, IFC, etc.) occurs while it is doing a serial poll. Since it already encountered one bus error it assumes that it cannot send the Serial Poll Disable (SPD) command. One rather common way of getting a bus error during a serial poll is to try to poll a device which is not connected to the bus. SPOLL will send out its talk address and wait for the response. None is forthcoming since the device isn't even there. Eventually a bus timeout error will occur (if the timeout value had been set to something other than 255) and SPOLL will return to your Basic program. But note that the devices on the 488 bus still think that a serial poll is in progress, and any device which is later addressed as a Talker will send its serial poll response byte instead of data. You can tell if this has occurred by checking the error code variable after the serial poll. If it shows a bus timeout error occurred the other devices on the bus think a serial poll

is still in progress. Your program will have to tell them that it is not. One way is to become a Controller and send Untalk (UNT is 5F Hex) followed by Serial Poll Disable (SPD is 19 Hex). Another way would be to send IFC (by calling function IFC), but this method may not be appropriate at times, because it resets all devices to their power-on state. You may not want to reprogram them.

**** How to Use MSOFT with Interpreter Basic ****

MSOFT.COM is comprised of two parts: one of which is temporary and is used for initialization, the second of which remains resident in your system until you exit from Basic (via the SYSTEM command). You must use a command line of the following form in order to bring in both MSOFT and Basic:

MSOFT MBASIC [filename options]

Notice that you may (but do not have to) specify the name of the Basic program which you want to run and you may also specify the normal Basic options, such as memory size, number of disk file buffers, etc. For example, if you want to run the Basic program BISAMPL.BAS and you also want to set the memory size option to limit Basic to only the first 32 Kbytes of memory, the command line would look like this:

MSOFT MBASIC BISAMPL /M:32767

Note that one and only one space must separate each of the commands on the command line.

**** NOTE ****

If you have renamed your copy of MBASIC to some new name, substitute the new name wherever "MBASIC" appears in these command line examples.

What MSOFT actually does is that it first relocates the resident module so that it lies just below the operating system (BDOS for CP/M). It then takes the rest of the command line and "submits" it to the operating system, just as if it were typed in by the user directly. This is the reason that you must give the name of your Basic interpreter on the command line. If you only type MSOFT on the command line MSOFT will relocate its resident module to lie just below the operating system and then return to the operating system. CP/M will then reload the CCP (Console Command Processor) to get your next command. However, the CCP also lies just below the operating system and destroys the resident module of MSOFT.

MSOFT also changes the JMP BDOS in location 0005H to a JMP to its own beginning address. That address contains a JMP BDOS so the BDOS calls (that is, CALL 0005H) work normally. MSOFT does all this to protect itself from the self-sizing feature of Basic.

**** How to Use MSOFT with Compiling Basic ****

The general scheme of operation is very similar to that used for the interpreter version of Basic, but there are a few differences. First and foremost is that the argument to a CALL in the interpreter must be an integer or integer variable. This is why each name ended with a percent sign (%). The compiler does NOT call an integer. Instead, the argument to its CALL is what is known as a PUBLIC LABEL. There are only two points that you really need to concern yourself with: (1) the name of each 488 function MUST BE the names shown earlier and (2) each name does NOT end with a percent sign. This means that while you may call the Serial Poll function any integer name you like in an interpreter program (SPOLL%, SP%, I%, etc.), you must call it SPOLL in a program to be compiled.

Since the argument of each CALL is a public label in compiling Basic, you do not have to tell MSOFT what variables to use (as is done in MBASINIT.BAS). Nor do you have to calculate SETUP%. It will not hurt anything if you do, but it is not necessary in a program which will be compiled. You cannot do a CALL SETUP (CNTL%, CNTLC%, ...) because the compiler will

not allow more than ten parameters to be passed through a CALL. There is no problem, because SETUP is not needed for the compiler anyway.

You do have to define all the communication variables (ERCODE%, TIME%, etc.) and CALL the functions IOSET and PROTCL. As in the interpreter version, you need to CALL ECHO only if you want to enable the input and/or output echo.

The program BCSAMPL.BAS is exactly the same as BISAMPL.BAS EXCEPT that the modifications necessary to make it compile have been made. Note that each 488 function name has had the ending percent sign (%) stripped off of it, since each is now a public label instead of an integer variable. Lines 1160 through 1300 have been removed.

The following dialog shows how the program BCSAMPL.BAS was compiled with Microsoft's version 5.30 Basic compiler and then linked to the MSOFT.REL file to generate the executable BCSAMPL.COM file. Note that BCSAMPL.COM stands alone: you need only type

BCSAMPL

to run it. This is in marked contrast to the interpreter version in which you have to type

MSOFT MBASIC BISAMPL

****** Example of how to Compile an MSOFT Program ******

NOTE: In the following dialog these conventions have been used:

1. Everything typed by the operator is shown underscored.
2. The character sequence <CR> means that the CARRIAGE RETURN key was typed. (Sometimes this key is labeled RETURN or ENTER.)
3. Version 5.30 of Microsoft's Basic compiler was used.
4. The "/O" switch was used so that BCSAMPL.COM will run without the BRUN.COM runtime package.

```
B>BASCOM<CR>
*=BCSAMPL/O<CR>

00000 Fatal Errors
14101 Bytes Free

B>L80<CR>

Link-80  3.43  14-Apr-81  Copyright (c) 1981 Microsoft

*BCSAMPL/E,BCSAMPL/N,MSOFT<CR>

Data      0103     4639     <17718>

18563 Bytes Free
[0162     4639     70]

B>
```

>>>>>>>>>>> NOTE <<<<<<<<<<<

The following source-code programs are included for illustrative purposes. Permission is granted to the reader to reproduce or abstract from these programs. These programs are the ONLY portion of this manual that may be reproduced without the prior written permission of

Pickles & Trout, P.O. Box 1206, Goleta, CA 93116

****

Comments on BISAMPL.BAS and BCSAMPL.BAS

These programs can be used to experiment with how MSOFT works, as well as experiment with how any device attached to the 488 bus responds to various commands. The programs differ only in that BISAMPL is the version that is for the interpreter version of Basic, while BCSAMPL is the version for the Basic compiler. They differ in that the arguments of all the CALLs in BISAMPL are integers (end with a % sign), while in BCSAMPL they are public labels. Also, BCSAMPL does not do a CALL SETUP(...).

These programs request the user to specify what function is to be performed by MSOFT, and whatever other information is needed in order to perform it. For instance, if the user indicates that the TALK function is to be used, the programs ask for the string that is to be sent over the 488 bus by the P&T-488 as a talker. After all necessary information has been entered the programs perform the function and report the value of the error code, what function was performed and any appropriate error message.

The programs have a special string collection routine (lines 3160 through 3410). Basic does not normally allow characters such as line feed and carriage return to be included in a string gathered from the console keyboard. However, it is often necessary to include these and other control characters in strings which are to be sent over the 488 bus while the P&T-488 is a talker or a controller. These control characters can be entered into the talk and control strings by preceding them with an ESCAPE character. For example, to get the string

1234<ESCAPE>$%<RETURN><LINE FEED>
you would type
1234<ESCAPE><ESCAPE>$%<ESCAPE><RETURN><ESCAPE><LINE FEED><RETURN>

Notice that each control code (<ESCAPE>, <RETURN> and <LINE FEED>) is preceded by an <ESCAPE>. The very last <RETURN> is not preceded by an <ESCAPE> because it is the delimiter telling Basic that the string is complete. The BACKSPACE key can be used to correct errors. BACKSPACE can be put into the string by preceding it (like the other control characters) with an <ESCAPE>. The only character that cannot be put into a string is Control C (ETX) because Basic recognizes it as an abort.

As an illustration of how to use these programs, assume you have a Hewlett-Packard 59309A Digital Clock. The programming codes for this clock are the following:

R Reset the clock to 01:01:00:00:00
P Stop the clock
T Start the clock
S Add one second to the time displayed by the clock
M Add one minute to the time displayed by the clock
H Add one hour
D Add one day

NOTE: When the front panel display is the following:

Month	Day	Hour	Minute	Second
12	28	11	23	14

The output to the 488 bus (when addressed to talk and with colon format) is in the following format:

(? or <SP> <SP>) : 12 : 28 : 11 : 23 : 14 <CR> <LF>
Status Space Month Day Hour Minute Second

The status character "?" means that there is an error. The status character <SP> means that there is no error.

The following example shows how to reset, set and read the time. It is assumed that the Talk address of the clock is "E" and the Listen address is "%". Underlined sections in the example are what the operator typed on the console: the rest is the computer's response. Note that the mnemonic <CR> means that the Carriage Return key is pressed, NOT that the four individual characters <, C, R, and > were pressed. To save paper [MENU] is shown in place of the menu that will actually appear on your console. The marginal comments indicate what it was that I was trying to accomplish at each step.

```
A>MSOFT MBASIC BISAMPL<CR>

P&T 488 - MBasic Interface Software  Revision 0.63
Copyright 1981,82 by Pickles & Trout

BASIC-80 Rev. 5.21
[CP/M Version]
Copyright 1977-81 (C) by Microsoft
Created: 28-Jul-81
24967 Bytes free


1.  CONTROL   Become the Controller and output a command string
2.  TALK      Become a Talker and send a string
3.  LISTEN    Become a Listener and receive a string
4.  REMOTE    Make the REN (Remote ENable) line true
5.  LOCAL     Make the REN line false
6.  IFC       Issue an IFC (InterFace Clear) command
7.  RESET     Reset the P&T 488 interface
8.  STATUS    Display the current 488 bus status
9.  SPOLL     Perform a Serial Poll of the 488 bus
10. PPOLL     Perform a Parallel Poll of the 488 bus
11. Change the communication protocol (EOT switch, EOS, and string length)
12. Change input echo, output echo and timeout values
13. Change S-100 port numbers (DIP switch on P&T-488 card must agree)

Which would you like to do? 6<CR>          send Interface Clear to 488 devices
Function = INTERFACE CLEAR  Error Code = 0
NORMAL RETURN                              no errors have occurred

[ MENU ]

Which would you like to do? 4<CR>          make REN line true
Function = REMOTE ENABLE    Error Code = 0
NORMAL RETURN
```

```
[ MENU ]

Which would you like to do? 1<CR>          address the clock as a Listener
Please enter the Control string
STRING: $<CR>
Function = CONTROLLER        Error Code = 2
NO LISTENERS - I cannot talk to myself!   Oh dear, I forgot to connect the 488 cable

[ MENU ]

Which would you like to do? 6<CR>          connected the cable, let's try again
Function = INTERFACE CLEAR  Error Code = 0
NORMAL RETURN

[ MENU ]

Which would you like to do? 4<CR>          make REN line true
Function = REMOTE ENABLE    Error Code = 0
NORMAL RETURN

[ MENU ]

Which would you like to do? 1<CR>          address the clock as a Listener
Please enter the Control string
STRING: $<CR>
Function = CONTROLLER        Error Code = 0
NORMAL RETURN                              and the clock's indicator shows that it is addressed

[ MENU ]

Which would you like to do? 2<CR>          set the clock to Jan 5, 8:10 AM and 15 seconds
Please enter the Talk string
STRING: RPDDDDHHHHHHHHMMMMMMMMSSSSSSSSSSSSSSS<CR>
Function = TALKER            Error Code = 0
NORMAL RETURN                              and the clock displays 01:05:08:10:15

[ MENU ]

Which would you like to do? 2<CR>          start the clock when the time is 8:10:15
Please enter the Talk string
STRING: T<CR>
Function = TALKER            Error Code = 0
NORMAL RETURN                              the clock is now running

[ MENU ]

Which would you like to do? 11<CR>         make line feed the EOS byte
The current communication protocol setup is:

    EOT switch      = 0
    EOS value       = 0
    String length   = 254

What is the new EOT switch? 1<CR>          terminate listen upon receipt of EOS
What is the new EOS value (0..255)? 10<CR>      make EOS a line feed
What is the new String Length (0..255)? 25<CR>  make maximum string 25 bytes
```

```
[ MENU ]

Which would you like to do? 1<CR>          unaddress clock as Listener, address as Talker
Please enter the Control string
STRING: ?E<CR>
Function = CONTROLLER        Error Code = 0
NORMAL RETURN

[ MENU ]

Which would you like to do? 3<CR>          listen to the clock
String heard on the 488 bus is:
  01:05:08:16:01                           that's the time!
Function = LISTENER          Error Code = 0
NORMAL RETURN

[ MENU ]

Which would you like to do? ↑C             press Control C to abort
Break in 1510
Ok
SYSTEM                                     exit Basic

B>
```

****** BISAMPL.BAS Listing ******

```
10 '
20 ' BISAMPL as of 9:54  4-8-82
30 '
40 '
1000 '  ================================================
1010 '
1020 '  Let the operator test each function and observe the
1030 '  response
1040 '
1050 '  Control characters (such as line feed and carriage return) can
1060 '  be entered into the TALK and CONTROL strings by preceding the
1070 '  control character with an ESCAPE.  For example, to get the string
1080 '  1234<ESCAPE>$$<RETURN><LINE FEED> you would type
1090 '  1234<ESCAPE><ESCAPE>$$<ESCAPE><RETURN><ESCAPE><LINE FEED><RETURN>.
1100 '
1110 ' ================================================
1120 '  Initialization Routines
1130 '
1140 '  The purpose of these routines is to initialize the MSOFT function
1150 '  addresses and the communication variables.
1160 '
1170 '  Calculate the address of the SETUP function
1180 '
1190 TEMP = 256*PEEK(7)+PEEK(6)+9
1200 IF TEMP>32767 THEN TEMP = TEMP-65536!
1210 SETUP% = CINT(TEMP)                    :'  convert it to an integer
1220 '
1230 '  Set up function call address variables
1240 '
1250 CALL SETUP% (CNTL%, CNTLC% ,TALK%, TALKC%, LSTN%, LSTNC%, SPOLL%, PPOLL%,
                  DREN%, REN%, STATUS%, IFC%, BRSET%, IOSET%, PROTCL%, ECHO%,
                  IOPORT%)
1260 '
1270 '  Call the setup routines to let MSOFT know what variables to use
1280 '
1290 CALL IOSET% (ERCODE%, TIME%, POLL%, BUS%)
1300 CALL PROTCL% (EOT%, EOS%, LENGTH%)
1310 CALL ECHO% (ECHOIN%, ECHOOUT%)
1320 '
1330 ' ================================================
1340 '  Main Menu
1350 '
1360 PRINT : PRINT
1370 PRINT "1.  CONTROL   Become the Controller and output a command string"
1380 PRINT "2.  TALK      Become a Talker and send a string"
1390 PRINT "3.  LISTEN    Become a Listener and receive a string"
1400 PRINT "4.  REMOTE    Make the REN (Remote ENable) line true"
1410 PRINT "5.  LOCAL     Make the REN line false"
1420 PRINT "6.  IFC       Issue an IFC (InterFace Clear) command"
1430 PRINT "7.  RESET     Reset the P&T 488 interface"
1440 PRINT "8.  STATUS    Display the current 488 bus status"
1450 PRINT "9.  SPOLL     Perform a Serial Poll of the 488 bus"
1460 PRINT "10. PPOLL     Perform a Parallel Poll of the 488 bus"
1470 PRINT "11. Change the communication protocol (EOT switch, EOS, and string length)"
1480 PRINT "12. Change input echo, output echo and timeout values"
1490 PRINT "13. Change S-100 port numbers (DIP switch on P&T-488 card must agree)"
```

```
1500 PRINT
1510 INPUT "Which would you like to do";F%  : '  get function code
1520 IF F%<1 OR F%>13 THEN PRINT CHR$(7):GOTO 1360
1530 PRINT
1540 ERCODE%=0                  : '  clear the error code
1550 '
1560 IF F%<>1 THEN 1630
1570 PRINT "Please enter the Control string"
1580 GOSUB 3200                 : '  get string to send as a controller
1590 FC$="CONTROLLER"
1600 CALL CNTL% (A1$)           : '  send out the command string
1610 GOTO 2730
1620 '
1630 IF F%<>2 THEN 1700
1640 PRINT "Please enter the Talk string"
1650 GOSUB 3200                 : '  get string to send as a talker
1660 FC$="TALKER"
1670 CALL TALK% (A1$)           : '  send out a data string
1680 GOTO 2730
1690 '
1700 IF F%<>3 THEN 1780
1710 A1$=""
1720 FC$="LISTENER"
1730 CALL LSTN%(A1$)            : '  get string from the 488
1740 PRINT "String heard on 488 bus is:"
1750 PRINT A1$
1760 GOTO 2730
1770 '
1780 IF F%<>4 THEN 1830
1790 FC$="REMOTE ENABLE"
1800 CALL REN%                  : '  make REN line true
1810 GOTO 2730
1820 '
1830 IF F%<>5 THEN 1880
1840 FC$="REMOTE DISABLE"
1850 CALL DREN%                 : '  make REN line false
1860 GOTO 2730
1870 '
1880 IF F%<>6 THEN 1930
1890 FC$="INTERFACE CLEAR"
1900 CALL IFC%                  : '  issue an IFC command
1910 GOTO 2730
1920 '
1930 IF F%<>7 THEN 1980
1940 FC$="RESET P&T 488"
1950 CALL BRSET%                : '  reset the P&T 488
1960 GOTO 2730
1970 '
1980 IF F%<>8 THEN 2040
1990 CALL STATUS%
2000 PRINT "Bus Status is: ";BUS%
2010 FC$="STATUS"
2020 GOTO 2730
2030 '
2040 IF F%<>9 THEN 2140
2050 PRINT "Please enter Talk addresses to poll"
2060 GOSUB 3200                 : '  get string of talk addresses
2070 PRINT
```

```
2080 FC$="SERIAL POLL"
2090 CALL SPOLL%(A1%,B1%)       : '  perform Serial Poll
2100 PRINT "Talk address of responding device is ";B1%
2110 PRINT "Poll response = ";POLL%
2120 GOTO 2730
2130 '
2140 IF F%<>10 THEN 2200
2150 FC$="PARALLEL POLL"
2160 CALL PPOLL%                : '  perform parallel poll
2170 PRINT "Poll response = ";POLL%
2180 GOTO 2730
2190 '
2200 IF F%<>11 THEN 2400
2210 PRINT:PRINT
2220 PRINT "The current communication protocol setup is:"
2230 PRINT
2240 PRINT "    EOT switch      = ";EOT%
2250 PRINT "    EOS value       = ";EOS%
2260 PRINT "    String length   = ";LENGTH%
2270 PRINT
2280 INPUT "What is the new EOT switch value";EOT%
2290 INPUT "What is the new EOS value (0..255)";EOS%
2300 IF EOS%>=0 AND EOS%<=255 THEN 2330
2310 PRINT "The EOS value must be between 0 and 255!"
2320 GOTO 2290
2330 INPUT "What is the new String Length (0..255)";LENGTH%
2340 IF LENGTH%>=0 AND LENGTH%<256 THEN 2370
2350 PRINT "The LENGTH must be between 0 and 255!"
2360 GOTO 2330
2370 PRINT
2380 GOTO 1360
2390 '
2400 IF F%<>12 THEN 2640
2410 PRINT:PRINT
2420 PRINT "The Input Echo, Output Echo, and Timeout are currently set to:"
2430 PRINT
2440 P$="N": IF ECHOIN%<>0 THEN P$="Y"
2450 PRINT "    Input Echo      ";P$
2460 P$="N": IF ECHOOUT%<>0 THEN P$="Y"
2470 PRINT "    Output Echo     ";P$
2480 PRINT "    Timeout Value   ";TIME%
2490 PRINT
2500 PRINT "Echo Input (Y/N)";
2510 INPUT A1$ : A1$=LEFT$(A1$,1)
2520 IF A1$<>"Y" AND A1$<>"N" AND A1$<>"y" AND A1$<>"n" THEN 2500
2530 ECHOIN%=0: IF A1$="Y" OR A1$="y" THEN ECHOIN%=1
2540 PRINT "Echo Output (Y/N)";
2550 INPUT A1$ : A1$=LEFT$(A1$,1)
2560 IF A1$<>"Y" AND A1$<>"N" AND A1$<>"y" AND A1$<>"n" THEN 2540
2570 ECHOOUT%=0: IF A1$="Y" OR A1$="y" THEN ECHOOUT%=1
2580 INPUT "What is the new TIMEOUT value (0..255)";TIME%
2590 IF TIME%>=0 AND TIME%<=255 THEN 2620
2600 PRINT "The TIMEOUT value must be between 0 and 255!"
2610 GOTO 2580
2620 PRINT
2630 GOTO 1360
2640 IF F%<>13 THEN 2730
2650 INPUT "What is the new S-100 port number (0..255)";PORT%
```

```
2660 PRINT
2670 CALL IOPORT$(PORT$)
2680 GOTO 1360
2690 '
2700 ' ============================================================
2710 '  Display function and error code, then return to main menu
2720 '
2730 PRINT
2740 GOSUB 2770          : '  print function and error message
2750 GOTO 1360           : ' go back to main menu
2760 '
2770 ' ============================================================
2780 '          Report 488 Function Errors
2790 '
2800 PRINT "Function = ";FC$;TAB(40);"Error Code = ";ERCODE%
2810 '
2820 ' Interpret Error codes and print error messages
2830 '
2840 IF ERCODE%<0 THEN 3140
2850 IF ERCODE%=0 THEN PRINT "NORMAL RETURN" : RETURN
2860 IF ERCODE%>255 THEN 3140
2870 FOR I=7 TO 0 STEP -1
2880 I0=2^I
2890 R9=ERCODE%-I0 : IF R9 < 0 THEN 3120
2900 ERCODE%=R9
2910 ON I+1 GOTO 2930,2960,2980,3010,3030,3060,3080,3100
2920 '
2930 PRINT "SETUP ERROR - either IOSET% or PROTCL% wasn't called before"
2940 PRINT "               using one of the MSOFT communication functions"
2950 GOTO 3120
2960 PRINT "NO LISTENERS - I cannot talk to myself!"
2970 GOTO 3120
2980 PRINT "SERIAL POLL ADDRESS ERROR - no more than one secondary address"
2990 PRINT "                            may follow a primary address"
3000 GOTO 3120
3010 PRINT "SERVICE REQUEST - a 488 device is requesting service"
3020 GOTO 3120
3030 PRINT "TIMEOUT ERROR - the specified amount of time has elapsed without"
3040 PRINT "                completing a 488 handshake cycle"
3050 GOTO 3120
3060 PRINT "ATN TRUE - an external controller is trying to issue a command"
3070 GOTO 3120
3080 PRINT "IFC TRUE - reset 488 interface"
3090 GOTO 3120
3100 PRINT "S-100 RESET - reset interface (use function 6 or 7)"
3110 GOTO 3120
3120 NEXT I
3130 RETURN
3140 PRINT "SYSTEM ERROR - an illegal error code has been encoutered"
3150 RETURN
3160 '
3170 ' ============================================================
3180 '                    String Input Routine
3190 '
3200 '  Get the string.  Gather control codes if preceded by <ESCAPE>.
3210 '
3220 A1$=""
3230 PRINT "STRING: ";
```

```
3240 A8$=INPUT$(1)
3250 IF ASC(A8$)<>13 THEN 3280  : '  <RETURN> terminates input
3260 PRINT
3270 RETURN
3280 '  Use backspace key for character at a time deletion
3290 IF ASC(A8$)=8 THEN IF LEN(A1$)>0 THEN 3310 ELSE 3240
3300 GOTO 3370
3310 A9$=RIGHT$(A1$,1)                    : '  keep deleted char
3320 A1$=LEFT$(A1$,LEN(A1$)-1)            : '  remove deleted char from string
3330 PRINT CHR$(8);" ";CHR$(8);           : ' delete char from CRT
3340 '  If deleted char is a control char must also delete leading caret
3350 IF ASC(A9$)<32 THEN PRINT CHR$(8);" ";CHR$(8);
3360 GOTO 3240
3370 IF ASC(A8$)=27 THEN A8$=INPUT$(1)  : '  <ESCAPE> means get next char
3380 '  Show the control character.  If not a space preceed character with
3390 '  a caret.  Change the control character into a printing character.
3400 IF A8$>=" " THEN PRINT A8$; ELSE PRINT "^"+CHR$(64+ASC(A8$));
3410 A1$=A1$+A8$    : '  Append the character to the string
3420 GOTO 3240
```

**** BCSAMPL.BAS ****

BCSAMPL performs the same function as BISAMPL, but is written for the Basic compiler. The differences between the two programs is a consequence of the difference between the interpreter and compiler versions of Microsoft Basic. You will notice that all arguments of CALLs in BCSAMPL are public labels, while in BISAMPL they are integers (end with a % sign). Also, lines 1170 through 1260 of BISAMPL are superfluous when the compiler is used, so they do not appear in BCSAMPL.

**** BCSAMPL.BAS Listing ****

```
10 '
20 ' BCSAMPL as of 10:32  4-8-82
30 '
40 '
1000 '  ====================================================
1010 '
1020 '  Let the operator test each function and observe the
1030 '  response
1040 '
1050 '  Control characters (such as line feed and carriage return) can
1060 '  be entered into the TALK and CONTROL strings by preceding the
1070 '  control character with an ESCAPE.  For example, to get the string
1080 '  1234<ESCAPE>$$<RETURN><LINE FEED> you would type
1090 '  1234<ESCAPE><ESCAPE>$$<ESCAPE><RETURN><ESCAPE><LINE FEED><RETURN>.
1100 '
1110 ' ====================================================
1120 '  Initialization Routines
1130 '
1140 '  The purpose of these routines is to initialize the MSOFT function
1150 '  addresses and the communication variables.
1160 '
1270 '  Call the setup routines to let MSOFT know what variables to use
1280 '
1290 CALL IOSET (ERCODE%, TIME%, POLL%, BUS%)
1300 CALL PROTCL (EOT%, EOS%, LENGTH%)
1310 CALL ECHO (ECHOIN%, ECHOOUT%)
1320 '
1330 ' ====================================================
1340 '  Main Menu
1350 '
1360 PRINT : PRINT
1370 PRINT "1.  CONTROL   Become the Controller and output a command string"
1380 PRINT "2.  TALK      Become a Talker and send a string"
1390 PRINT "3.  LISTEN    Become a Listener and receive a string"
1400 PRINT "4.  REMOTE    Make the REN (Remote ENable) line true"
1410 PRINT "5.  LOCAL     Make the REN line false"
1420 PRINT "6.  IFC       Issue an IFC (interface Clear) command"
1430 PRINT "7.  RESET     Reset the P&T 488 interface"
1440 PRINT "8.  STATUS    Display the current 488 bus status"
1450 PRINT "9.  SPOLL     Perform a Serial Poll of the 488 bus"
1460 PRINT "10. PPOLL     Perform a Parallel Poll of the 488 bus"
1470 PRINT "11. Change the communication protocol (EOT switch, EOS, and string length)"
1480 PRINT "12. Change input echo, output echo and timeout values"
1490 PRINT "13. Change S-100 port numbers (DIP switch on P&T-488 card must agree)"
1500 PRINT
```

```
1510 INPUT "Which would you like to do";F%  : '  get function code
1520 IF F%<1 OR F%>13 THEN PRINT CHR$(7):GOTO 1360
1530 PRINT
1540 ERCODE%=0                  : '  clear the error code
1550 '
1560 IF F%<>1 THEN 1630
1570 PRINT "Please enter the Control string"
1580 GOSUB 3200                 : '  get string to send as a controller
1590 FC$="CONTROLLER"
1600 CALL CNTL (A1$)            : '  send out the command string
1610 GOTO 2730
1620 '
1630 IF F%<>2 THEN 1700
1640 PRINT "Please enter the Talk string"
1650 GOSUB 3200                 : '  get string to send as a talker
1660 FC$="TALKER"
1670 CALL TALK (A1$)            : '  send out a data string
1680 GOTO 2730
1690 '
1700 IF F%<>3 THEN 1780
1710 A1$=""
1720 FC$="LISTENER"
1730 CALL LSTN(A1$)             : '  get string from the 488
1740 PRINT "String heard on 488 bus is:"
1750 PRINT A1$
1760 GOTO 2730
1770 '
1780 IF F%<>4 THEN 1830
1790 FC$="REMOTE ENABLE"
1800 CALL REN                   : '  make REN line true
1810 GOTO 2730
1820 '
1830 IF F%<>5 THEN 1880
1840 FC$="REMOTE DISABLE"
1850 CALL DREN                  : '  make REN line false
1860 GOTO 2730
1870 '
1880 IF F%<>6 THEN 1930
1890 FC$="INTERFACE CLEAR"
1900 CALL IFC                   : '  issue an IFC command
1910 GOTO 2730
1920 '
1930 IF F%<>7 THEN 1980
1940 FC$="RESET P&T 488"
1950 CALL BRSET                 : '  reset the P&T 488
1960 GOTO 2730
1970 '
1980 IF F%<>8 THEN 2040
1990 CALL STATUS
2000 PRINT "Bus Status is: ";BUS%
2010 FC$="STATUS"
2020 GOTO 2730
2030 '
2040 IF F%<>9 THEN 2140
2050 PRINT "Please enter Talk addresses to poll"
2060 GOSUB 3200                 : '  get string of talk addresses
2070 PRINT
2080 FC$="SERIAL POLL"
```

```
2090 CALL SPOLL(A1$,B1$)        : '  perform Serial Poll
2100 PRINT "Talk address of responding device is ";B1$
2110 PRINT "Poll response = ";POLL$
2120 GOTO 2730
2130 '
2140 IF F$<>10 THEN 2200
2150 FC$="PARALLEL POLL"
2160 CALL PPOLL                 : '  perform parallel poll
2170 PRINT "Poll response = ";POLL$
2180 GOTO 2730
2190 '
2200 IF F$<>11 THEN 2400
2210 PRINT:PRINT
2220 PRINT "The current communication protocol setup is:"
2230 PRINT
2240 PRINT "    EOT switch      = ";EOT$
2250 PRINT "    EOS value       = ";EOS$
2260 PRINT "    String length   = ";LENGTH$
2270 PRINT
2280 INPUT "What is the new EOT switch value";EOT$
2290 INPUT "What is the new EOS value (0..255)";EOS$
2300 IF EOS$>=0 AND EOS$<=255 THEN 2330
2310 PRINT "The EOS value must be between 0 and 255!"
2320 GOTO 2290
2330 INPUT "What is the new String Length (0..255)";LENGTH$
2340 IF LENGTH$>=0 AND LENGTH$<256 THEN 2370
2350 PRINT "The LENGTH must be between 0 and 255!"
2360 GOTO 2330
2370 PRINT
2380 GOTO 1360
2390 '
2400 IF F$<>12 THEN 2640
2410 PRINT:PRINT
2420 PRINT "The Input Echo, Output Echo, and Timeout are currently set to:"
2430 PRINT
2440 P$="N": IF ECHOIN$<>0 THEN P$="Y"
2450 PRINT "    Input Echo      ";P$
2460 P$="N": IF ECHOOUT$<>0 THEN P$="Y"
2470 PRINT "    Output Echo     ";P$
2480 PRINT "    Timeout Value   ";TIME$
2490 PRINT
2500 PRINT "Echo Input (Y/N)";
2510 INPUT A1$ : A1$=LEFT$(A1$,1)
2520 IF A1$<>"Y" AND A1$<>"N" AND A1$<>"y" AND A1$<>"n" THEN 2500
2530 ECHOIN$=0: IF A1$="Y" OR A1$="y" THEN ECHOIN$=1
2540 PRINT "Echo Output (Y/N)";
2550 INPUT A1$ : A1$=LEFT$(A1$,1)
2560 IF A1$<>"Y" AND A1$<>"N" AND A1$<>"y" AND A1$<>"n" THEN 2540
2570 ECHOOUT$=0: IF A1$="Y" OR A1$="y" THEN ECHOOUT$=1
2580 INPUT "What is the new TIMEOUT value (0..255)";TIME$
2590 IF TIME$>=0 AND TIME$<=255 THEN 2620
2600 PRINT "The TIMEOUT value must be between 0 and 255!"
2610 GOTO 2580
2620 PRINT
2630 GOTO 1360
2640 IF F$<>13 THEN 2730
2650 INPUT "What is the new S-100 port number (0..255)";PORT$
2660 PRINT
```

```
2670 CALL IOPORT(PORT%)
2680 GOTO 1360
2690 '
2700 ' ============================================================
2710 '  Display function and error code, then return to main menu
2720 '
2730 PRINT
2740 GOSUB 2770          : '  print function and error message
2750 GOTO 1360           : ' go back to main menu
2760 '
2770 ' ============================================================
2780 '          Report 488 Function Errors
2790 '
2800 PRINT "Function = ";FC$;TAB(40);"Error Code = ";ERCODE%
2810 '
2820 ' interpret Error codes and print error messages
2830 '
2840 IF ERCODE%<0 THEN 3140
2850 IF ERCODE%=0 THEN PRINT "NORMAL RETURN" : RETURN
2860 IF ERCODE%>255 THEN 3140
2870 FOR I=7 TO 0 STEP -1
2880 I0=2^I
2890 R9=ERCODE%-I0 : IF R9 < 0 THEN 3120
2900 ERCODE%=R9
2910 ON I+1 GOTO 2930,2960,2980,3010,3030,3060,3080,3100
2920 '
2930 PRINT "SETUP ERROR - either IOSET% or PROTCL% wasn't called before"
2940 PRINT "               using one of the MSOFT communication functions"
2950 GOTO 3120
2960 PRINT "NO LISTENERS - I cannot talk to myself!"
2970 GOTO 3120
2980 PRINT "SERIAL POLL ADDRESS ERROR - no more than one secondary address"
2990 PRINT "                            may follow a primary address"
3000 GOTO 3120
3010 PRINT "SERVICE REQUEST - a 488 device is requesting service"
3020 GOTO 3120
3030 PRINT "TIMEOUT ERROR - the specified amount of time has elapsed without"
3040 PRINT "                completing a 488 handshake cycle"
3050 GOTO 3120
3060 PRINT "ATN TRUE - an external controller is trying to issue a command"
3070 GOTO 3120
3080 PRINT "IFC TRUE - reset 488 interface"
3090 GOTO 3120
3100 PRINT "S-100 RESET - reset interface (use function 6 or 7)"
3110 GOTO 3120
3120 NEXT I
3130 RETURN
3140 PRINT "SYSTEM ERROR - an illegal error code has been encoutered"
3150 RETURN
3160 '
3170 ' ============================================================
3180 '                      String Input Routine
3190 '
3200 '  Get the string.  Gather control codes if preceded by <ESCAPE>.
3210 '
3220 A1$=""
3230 PRINT "STRING: ";
3240 A8$=INPUT$(1)
```

```
3250 IF ASC(A8$)<>13 THEN 3280  : '  <RETURN> terminates input
3260 PRINT
3270 RETURN
3280 '  Use backspace key for character at a time deletion
3290 IF ASC(A8$)=8 THEN IF LEN(A1$)>0 THEN 3310 ELSE 3240
3300 GOTO 3370
3310 A9$=RIGHT$(A1$,1)                  : '  keep deleted char
3320 A1$=LEFT$(A1$,LEN(A1$)-1)          : '  remove deleted char from string
3330 PRINT CHR$(8);" ";CHR$(8);         : ' delete char from CRT
3340 '  If deleted char is a control char must also delete leading caret
3350 IF ASC(A9$)<32 THEN PRINT CHR$(8);" ";CHR$(8);
3360 GOTO 3240
3370 IF ASC(A8$)=27 THEN A8$=INPUT$(1)  : '  <ESCAPE> means get next char
3380 '  Show the control character.  If not a space preceed character with
3390 '  a caret.  Change the control character into a printing character.
3400 IF A8$>=" " THEN PRINT A8$; ELSE PRINT "^"+CHR$(64+ASC(A8$));
3410 A1$=A1$+A8$    : '  Append the character to the string
3420 GOTO 3240
```

**** B488INIT.BAS ****

This program fragment is included on your disk as an aid in writing programs for MSOFT. All of the setup calls are included. Its primary utility is that all the variables are called in the correct order in the setup routines. Remember that Basic does not check to make sure that the right number of parameters are passed, nor does it check to make sure they are of the correct type. Since B488INIT.BAS has all of the setup calls in it, if you copy it to your program you are sure that the right number and type of parameters are used. Also, you are spared the frustration of spending hours trying to get a program to work only to find out that you have mispelled a function name, or have accidently changed the order of the parameters.

**** B488INIT.BAS Listing ****

```
100 ' ============================================================
110 '  Initialization Routines
120 '
130 '  The purpose of these routines is to initialize the MBAS488 function
140 '  addresses and the communication variables.
150 '
160 '  Calculate the address of the SETUP function
170 '
180 TEMP = 256*PEEK(7)+PEEK(6)+9
190 IF TEMP>32767 THEN TEMP = TEMP-65536!
200 SETUP% = CINT(TEMP)              :'  convert it to an integer
210 '
220 '  Set up function call address variables
230 '
240 CALL SETUP% (CNTL%, CNTLC% ,TALK%, TALKC%, LSTN%, LSTNC%, SPOLL%, PPOLL%,
                 DREN%, REN%, STATUS%, IFC%, BRSET%, IOSET%, PROTCL%, ECHO%,
                 IOPORT%)
250 '
260 '  Call the setup routines to let MBAS488 know what variables to use
270 '
280 CALL IOSET% (ERCODE%, TIME%, POLL%, BUS%)
290 CALL PROTCL% (EOT%, EOS%, LENGTH%)
300 CALL ECHO% (ECHOIN%, ECHOOUT%)
310 '
320 ' ============================================================
```

**** BICLOCK.BAS ****

This program demonstrates how simple an interpreter Basic program can be. The first part is a copy of B488INIT, and the error-reporting subroutine was lifted from BISAMPL. Thus only lines 1340 through 1650 are unique to this program. This program initializes the 488 bus (by sending an Interface Clear), puts an HP 59309 clock into the Remote mode (by making the REN line true and then sending the clock's Listen Address). It then addresses the clock as a Talker and listens to the data (status, date and time) that the clock sends over the bus. It displays the date and time each time the minutes change. It also displays the data each time the status character indicates a clock error.

```
10 '
20 ' BICLOCK as of 14:30  4-09-82
30 '
40 '
1000 '  ================================================
1010 '
1020 '  This is an interpreter Basic program which addresses an
1030 '  HP 59309A clock as a talker and then reads the time and
1040 '  date.  It continually rereads the time and displays the
1050 '  time and date on the console each minute.
1060 '
1070 '  The program assumes that the bus output format of the
1080 '  59309A is set to SPACE, CAL and COLON.  It also assumes
1090 '  that the TALK address of the clock is "E" and the
1100 '  LISTEN address of the clock is "%".
1110 '
1120 ' ================================================
1130 '  Initialization Routines
1140 '
1150 '  The purpose of these routines is to initialize the MSOFT function
1160 '  addresses and the communication variables.
1170 '
1180 '  Calculate the address of the SETUP function
1190 '
1200 TEMP = 256*PEEK(7)+PEEK(6)+9
1210 IF TEMP>32767 THEN TEMP = TEMP-65536!
1220 SETUP% = CINT(TEMP)                    :'  convert it to an integer
1230 '
1240 '  Set up function call address variables
1250 '
1260 CALL SETUP% (CNTL%, CNTLC% ,TALK%, TALKC%, LSTN%, LSTNC%, SPOLL%, PPOLL%,
                  DREN%, REN%, STATUS%, IFC%, BRSET%, IOSET%, PROTCL%, ECHO%,
                  IOPORT%)
1270 '
1280 '  Call the setup routines to let MSOFT know what variables to use
1290 '
1300 CALL IOSET% (ERCODE%, TIME%, POLL%, BUS%)
1310 CALL PROTCL% (EOT%, EOS%, LENGTH%)
1320 CALL ECHO% (ECHOIN%, ECHOOUT%)
1330 '
1340 CALL IFC%                  :'  Do an Interface Clear (IFC)
1350 CALL REN%                  :'  Make the REN line true
1360 A1$="?"+CHR$(95)+"%"       :'  Unlisten, Untalk, Listen Address "%"
1370 CALL CNTLC%(A1$)           :'  Become the Controller and output A1$
1380 '                              (This puts the clock into the REMOTE mode)
1390 IF ERCODE%<>0 THEN 1640    :'  Report any errors
1400 '
```

```
1410 TIME%=255                  :' Do not time handshake
1420 EOT%=1                     :' Stop on End-Of-String (EOS) byte
1430 EOS%=10                    :' Make line feed the EOS byte
1440 OLDMIN%=-1                 :' Make OLDMIN some value which cannot
1450 '                                 match a clock reading
1460 '
1470 A1$="?"+CHR$(95)+"E"       :' Unlisten, Untalk, Talk Address "E"
1480 CALL CNTLC%(A1$)           :' Become the Controller and output A1$
1490 IF ERCODE%<>0 THEN 1640    :' Report any errors
1500 CALL LSTNC%(A2$)           :' Read the clock
1510 IF ERCODE%<>0 THEN 1640    :' Report any errors
1520 ' If the first character is a "?" then the clock is in error
1530 IF MID$(A2$,1,1)=" " THEN 1580
1540 PRINT "CLOCK ERROR  ";A2$
1550 PRINT "Reset clock"        :' Tell operator clock needs resetting
1560 END                        :' Then exit program
1570 ' Make MIN% the value of the unit minutes character
1580 MIN%=ASC(MID$(A2$,13,1))
1590 ' Show the time if the minutes have changed
1600 IF MIN%<>OLDMIN% THEN PRINT A2$
1610 OLDMIN%=MIN%               :' Update OLDMIN%
1620 GOTO 1460                  :' Read the clock again
1630 '
1640 GOSUB 1670                 :' Report the error
1650 GOTO 1340                  :' go back to IFC, REN, etc
1660 '
1670 ' ==========================================================
1680 '          Report 488 Function Errors
1690 '
1700 ' Interpret Error codes and print error messages
1710 '
1720 IF ERCODE%<0 THEN 2020
1730 IF ERCODE%=0 THEN RETURN
1740 IF ERCODE%>255 THEN 2020
1750 FOR I=7 TO 0 STEP -1
1760 I0=2^I
1770 R9=ERCODE%-I0 : IF R9 < 0 THEN 2000
1780 ERCODE%=R9
1790 ON I+1 GOTO 1810,1840,1860,1890,1910,1940,1960,1980
1800 '
1810 PRINT "SETUP ERROR - either IOSET% or PROTCL% wasn't called before"
1820 PRINT "               using one of the MSOFT communication functions"
1830 GOTO 2000
1840 PRINT "NO LISTENERS - I cannot talk to myself!"
1850 GOTO 2000
1860 PRINT "SERIAL POLL ADDRESS ERROR - no more than one secondary address"
1870 PRINT "                            may follow a primary address"
1880 GOTO 2000
1890 PRINT "SERVICE REQUEST - a 488 device is requesting service"
1900 GOTO 2000
1910 PRINT "TIMEOUT ERROR - the specified amount of time has elapsed without"
1920 PRINT "                completing a 488 handshake cycle"
1930 GOTO 2000
1940 PRINT "ATN TRUE - an external controller is trying to issue a command"
1950 GOTO 2000
1960 PRINT "IFC TRUE - reset 488 interface"
1970 GOTO 2000
1980 PRINT "S-100 RESET"
```

```
1990 GOTO 2000
2000 NEXT I
2010 RETURN
2020 PRINT "SYSTEM ERROR - an illegal error code has been encountered"
2030 RETURN
```

**** Parameter Passing ****

Even though MSOFT is designed to work with Microsoft Basic, it can be used with some other languages as well. Programs written in assembler, C, Microsoft Fortran and Pascal MT+ are shown in the following pages to demonstrate how MSOFT can be used with these languages.

Most languages require some assembler code in order to convert the Microsoft Basic parameter passing convention into whatever the language requires. For instance, C passes parameters on the stack but the called routine must not remove them from the stack. Pascal MT+ passes parameters on the stack and requires that the called routine remove them from the stack. In general, each language is slightly different and will require different parameter-passing conversion programs.

MSOFT is designed to interface with Microsoft Basic, so it uses exactly the same method of passing parameters as Microsoft Basic. The instruction

CALL PGM(param1, param2, param3, param4)

passes the four parameters param1, param2, param3 and param4 to PGM. Unlike most modern languages, Basic passes parameters by reference instead of by value. What this means is that Basic passes the address of the parameter to the called program.

MSOFT uses only two kinds of parameters: integers and strings. Basic stores integers as 16 bit (two byte) quantities, with the low order byte stored at the address, and the high order byte at location address+1. Basic stores strings in two parts: one is the string itself, and the other part is the string descriptor (sometimes called a dope vector). The string itself is stored in contiguous memory locations, with the leftmost character stored in the lowest address. The string descriptor is a three-byte block. The first byte contains the number of characters in the string (0..255), and the remaining two bytes contain the address of the first character of the string. As usual, Basic stores the address low order byte first.

When Basic passes an integer parameter, it actually passes the 16 bit address where that integer is stored. And when Basic passes a string, it actually passes the address of the descriptor block of that string. The called program has to look in that descriptor block to find the actual address of the string.

Basic does not indicate in any manner whatsoever the number or type of arguments passed. This is why it is so important that you make sure that the number and type are correct when you write a program. There is no way for the called programs to check for correctness of number and type.

Basic passes the parameters in the 16 bit registers of the 8080/Z-80, and, if there are more than three parameters, in a parameter table. The address of the first parameter is passed in register pair HL. The address of the second parameter (if any) is passed in register pair DE. If there are three parameters, the address of the third parameter is passed in register pair BC. If there are more than three parameters, the third through last parameters are put into a table and register pair BC contains the beginning address of that table. The table is organized as low byte of parameter 3, high byte of parameter 3, low byte of parameter 4, etc.

Let us look at a few examples.

```
CALL ECHO(ECHOIN$, ECHOUT$)
    HL = address of ECHOIN$
    DE = address of ECHOUT$
```

```
CALL TALK (A$)
    HL = address of descriptor block for A$
        A$ descriptor block:
        Byte 0 = string length
        Byte 1 = low byte of string address
        Byte 2 = high byte of string address

CALL IOSET (ERCODE%, TIME%, POLL%, BUS%)
    HL = address of ERCODE%
    DE = address of TIME%
    BC = address of parameter table
        parameter table:
        Byte 0 = low byte of address of POLL%
        Byte 1 = high byte of address of POLL%
        Byte 2 = low byte of address of BUS%
        Byte 3 = high byte of address of BUS%
```

****** CLOCK.MAC ******

This program performs the same function as the Basic program BICLOCK, but this one is written in 8080 assembler. Notice how the addresses of the parameters are placed in the registers before the MSOFT functions are called.

The following dialog shows how to assemble and link this program with MSOFT.REL. The result is an executable file named CLOCK.COM.

```
B>M80 =CLOCK<CR>

No  Fatal error(s)

B>L80 CLOCK/E,CLOCK/N,MSOFT<CR>

Link 80   3.42   19-Feb-81  Copyright (c) 1981 Microsoft

Data      0103   0B2B   < 2600>

38769 Bytes Free
[0000     0B2B    11]

B>
```

```
;       CLOCK.MAC       4-12-82  15:32
;
; This assembly program is designed to be used with the MSOFT interface
; software for the P&T-488.  The primary purpose of this program is to
; illustrate how one can use MSOFT from an assembly program.
;
; This program first initializes the 488 bus by sending an Interface Clear.
; it then puts an HP 59309A clock into the remote mode by making the REN
; line true and then addressing the clock as a Listener.  This program
; then addresses the clock as a Talker and listens to the data (status,
; date and time) that the clock sends over the bus.  It displays the date
; and time each time the minutes change.  It also displays the data each
; time the status character indicates a clock error.
;
; The program assumes that the bus output format of the 59309A is set
```

```
;  to SPACE, CAL and COLON.  It also assumes that the TALK address of the
;  clock is "E" and the LISTEN address is "5".
;
;  Declare MSOFT routines as EXTernal references
;
        EXT     CNTL, CNTLC, TALK, TALKC, LSTN, LSTNC
        EXT     SPOLL, PPOLL, DREN, REN, STATUS, IFC
        EXT     BRSET, IOSET, PROTCL, ECHO, IOPORT
;
CR      EQU     13      ;ASCII carriage return
LF      EQU     10      ;ASCII line feed
ES      EQU     '$'     ;CP/M end of string character
BOOT    EQU     0       ;CP/M reboot entry
BDOS    EQU     5       ;standard CP/M entry
;
        JMP     CLOCK   ;jump to beginning of the program
;
ERCODE: DW      0       ;storage area for 488 error code
TIME:   DW      0       ;storage area for 488 timeout
EOT:    DW      0       ;storage area for 488 EOT switch
EOS:    DW      0       ;storage area for 488 EOS byte
LENGTH: DW      0       ;storage area for 488 listen string length
POLL:   DW      0       ;storage area for 488 poll response
ECHOIN: DW      0       ;storage area for 488 input echo switch
ECHOUT: DW      0       ;storage area for 488 output echo switch
BUS:    DW      0       ;storage area for 488 bus status variable
;  B register buffer
BBFR1:  DW      0
BBFR2:  DW      0
;
OLDMIN: DB      0       ;previous minutes reading
STRVCR: DS      3       ;string vector (count followed by address)
STRBFR: DS      64      ;a string buffer
;
        DS      32      ;stack area
STAK:
;
CLKMSG: DB      CR,LF,'CLOCK ERROR    ',ES
RSTMSG: DB      'Reset clock',CR,LF,ES
ERMSG1: DB      CR,LF,'SETUP ERROR - either IOSET or PROTCL was not '
        DB      'called before using'
        DB      CR,LF,'one of the MSOFT communication functions',CR,LF,ES
ERMSG2: DB      CR,LF,'NO LISTENERS - I cannot talk to myself!',CR,LF,ES
ERMSG3: DB      CR,LF,'SERIAL POLL ADDRESS ERROR - no more than one secondary'
        DB      CR,LF,'address may follow a primary address',CR,LF,ES
ERMSG4: DB      CR,LF,'SERVICE REQUEST - a 488 device is requesting service'
        DB      CR,LF,ES
ERMSG5: DB      CR,LF,'TIMEOUT ERROR - the specified amount of time has elapsed'
        DB      CR,LF,'without completing a 488 handshake cycle',CR,LF,ES
ERMSG6: DB      CR,LF,'ATN TRUE - an external controller is trying to issue'
        DB      ' a command',CR,LF,ES
ERMSG7: DB      CR,LF,'IFC TRUE - reset 488 interface',CR,LF,ES
ERMSG8: DB      CR,LF,'S-100 RESET',CR,LF,ES
;
CLOCK:  LXI     H,BUS
        SHLD    BBFR2   ;save address in second entry of B reg buffer
        LXI     H,POLL
        SHLD    BBFR1
```

```
        LXI     B,BBFR1 ;point BC to B register buffer
        LXI     D,TIME  ;point DE to address of word holding TIME
        LXI     H,ERCODE
        CALL    IOSET
;
        LXI     B,LENGTH
        LXI     D,EOS
        LXI     H,EOT
        CALL    PROTCL
;
        LXI     D,ECHOUT
        LXI     H,ECHOIN
        CALL    ECHO
;
;  Issue an IFC command
        CALL    IFC
;
;  Make REN line true
        CALL    REN
;
;  TIME contains the amount of time to allow for handshake.
;  If TIME=255, then the handshake is not timed.
        LXI     H,255
        SHLD    TIME
;
;  Turn off input and output echo
        LXI     H,0
        SHLD    ECHOIN
        SHLD    ECHOUT
;
;  Set up MSOFT so that it will stop on EOS (End-Of-String) byte,
;  set the EOS byte to be a line feed.
        LXI     H,1
        SHLD    EOT
        LXI     H,10
        SHLD    EOS
;
;  Set up a string for the Control function.  We will make the string
;  three bytes long: UNLISTEN, UNTALK and LAD (Listen Address of the clock)
        LXI     H,STRVCR ;point HL to the string descriptor vector
        LXI     D,STRBFR ;point DE to the string buffer
        MVI     M,3      ;put the count in the first byte of the vector
        INX     H
        MOV     M,E      ;put the address of the string in the next word
        INX     H        ;  of the vector
        MOV     M,D
;  Now put the characters into the string
        MVI     A,'?'    ;UNLISTEN
        STAX    D
        INX     D
        MVI     A,'_'    ;UNTALK
        STAX    D
        INX     D
        MVI     A,'%'    ;LAD (Listen Address of the clock)
        STAX    D
;
;  Now send the string over the 488 bus as a controller
        LXI     H,STRVCR
```

```
            CALL    CNTLC
      ;
      ;  Check error code and report any errors
            CALL    ERRCHK
      ;
      ;  If there is a bus error start the program again
            LDA     ERCODE
            ORA     A
            JNZ     CLOCK
      ;
      ;  Set up a string for the Control function.  We will make the string
      ;  three bytes long: UNLISTEN, UNTALK and TAD (Talk Address of the clock)
     REDTIM: LXI    H,STRVCR ;point HL to the string descriptor vector
            LXI     D,STRBFR ;point DE to the string buffer
            MVI     M,3      ;put the count in the first byte of the vector
            INX     H
            MOV     M,E      ;put the address of the string in the next word
            INX     H        ;  of the vector
            MOV     M,D
     ;  Now put the characters into the string
            MVI     A,'?'    ;UNLISTEN
            STAX    D
            INX     D
            MVI     A,'_'    ;UNTALK
            STAX    D
            INX     D
            MVI     A,'E'    ;TAD (Talk Address of the clock)
            STAX    D
      ;
      ;  Now send the string over the 488 bus as a controller
            LXI     H,STRVCR
            CALL    CNTLC
      ;
      ;  Check error code and report any errors
            CALL    ERRCHK
      ;
      ;  If there is a bus error start the program again
            LDA     ERCODE
            ORA     A
            JNZ     CLOCK
      ;
      ;  Now become a listener and read the time from the clock
            LXI     H,STRVCR ;tell LSTNC where the string vector is kept
            CALL    LSTNC    ;listen to the clock
      ;
      ;  Check error code and report any errors
            CALL    ERRCHK
      ;
      ;  If there is a bus error start the program again
            LDA     ERCODE
            ORA     A
            JNZ     CLOCK
      ;
      ;  No 488 bus error, so look at the string we got from the clock
            LXI     H,STRVCR ;point HL to the string vector again
            MOV     C,M      ;C=count (length of string)
            INX     H        ;point to the address of the string
            MOV     E,M
```

```
        INX     H
        MOV     D,M     ;DE=address of string heard on the 488 bus
;
;  Look at clock status byte to see if there is a problem
        LDAX    D
        CPI     '?'
        JZ      CLKERR  ;..clock error, so report it
;
;  See if the minutes have changed since the last time the clock was read
        LXI     H,12    ;units digit of minutes is the 13th byte of
                        ;  the string
        DAD     D       ;HL now points to the units digit of the minutes
        LDA     OLDMIN  ;get old value of units digit of minutes
        CMP     M       ;compare it to the new value
        MOV     A,M     ;update the old value for the next time
        STA     OLDMIN
        CNZ     SHOTIM  ;..display the time if units digit has changed
        JMP     REDTIM  ;read the time again
;
;  This subroutine reports any MSOFT errors on the console
ERRCHK: LDA     ERCODE  ;get the error code
        RAR             ;rotate right
        LXI     D,ERMSG1 ;DE points to appropriate error message
        CC      SHOERR   ;if carry set, display the error message
        RAR              ;rotate right
        LXI     D,ERMSG2 ;DE points to appropriate error message
        CC      SHOERR   ;if carry set, display the error message
        RAR              ;rotate right
        LXI     D,ERMSG3 ;DE points to appropriate error message
        CC      SHOERR   ;if carry set, display the error message
        RAR              ;rotate right
        LXI     D,ERMSG4 ;DE points to appropriate error message
        CC      SHOERR   ;if carry set, display the error message
        RAR              ;rotate right
        LXI     D,ERMSG5 ;DE points to appropriate error message
        CC      SHOERR   ;if carry set, display the error message
        RAR              ;rotate right
        LXI     D,ERMSG6 ;DE points to appropriate error message
        CC      SHOERR   ;if carry set, display the error message
        RAR              ;rotate right
        LXI     D,ERMSG7 ;DE points to appropriate error message
        CC      SHOERR   ;if carry set, display the error message
        RAR              ;rotate right
        LXI     D,ERMSG8 ;DE points to appropriate error message
        CC      SHOERR   ;if carry set, display the error message
        RET
;
;  This subroutine displays the clock error message and the time
;  read from the clock on the console.  It then jumps back to the
;  read time routine.
CLKERR: PUSH    B       ;save string length counter
        PUSH    D       ;save pointer to listen string
        LXI     D,CLKMSG ;point to clock error message
        CALL    SHOERR  ;display it on the console
        POP     D       ;DE points to beginning of listen string again
        POP     B       ;C contains the string length
        CALL    SHOTIM  ;display the string we got from the clock
        LXI     D,RSTMSG ;point to reset message
```

```
        CALL    SHOERR  ;and display it on the console
        JMP     BOOT    ;and go to operating system
;
;  This subroutine displays the message pointed to by DE on the console
SHOERR: PUSH    PSW     ;preseve flags and reg A
        MVI     C,9     ;select print string function
        CALL    BDOS
        POP     PSW     ;restore flags and reg A
        RET
;
;  This subroutine displays the time on the system console.  It uses the
;  unbuffered CP/M console output function.
SHOTIM: SUB     A       ;clear reg A
        ORA     C       ;see if count is zero
        RZ              ;..count is zero, so do not print anything
SHOT1:  LDAX    D       ;get the character
        INX     D       ;point to next
        PUSH    D       ;preserve pointer from damage by CP/M
        PUSH    B       ;preserve counter from damage by CP/M
        MOV     E,A     ;put the character in reg E as needed by CP/M
        MVI     C,2     ;select console output function
        CALL    BDOS
        POP     B       ;get character counter again
        POP     D       ;get character pointer again
        DCR     C       ;decrement the count
        JNZ     SHOT1   ;..loop until all characters printed
        MVI     E,LF    ;finish with a line feed
        MVI     C,2
        CALL    BDOS
        RET

        END
```

****** MTSAMPL.PAS ******

This program performs the same functions as BISAMPL.BAS and BCSAMPL.BAS. It is written in Pascal MT+ (a product of MT Microsystems, Inc.). It requires the program MT488.MAC, which is an assembler program which performs the necessary parameter passing conversions. The listing of MT488.MAC follows MTSAMPL.PAS.

The major difference between MTSAMPL and BISAMPL is that MTSAMPL has a 14th menu item, namely, the option of returning to the operating system. This option was not needed in BISAMPL or BCSAMPL since Microsoft Basic will abort a program when it detects a Control C typed on the console.

One point that you should notice is that all of the formal parameters of the MSOFT functions (external procedures pcntl through ploprt) are variable parameters (denoted by var). Pascal passes variable parameters by reference instead of by value. This means that Pascal will pass to MT488 (and thence to MSOFT) the addresses of the parameters instead of the values. MSOFT requires the addresses, so remember that you must declare the parameters of the MSOFT functions to be variable parameters.

The following dialog shows how to compile the program MTSAMPL.PAS, assemble MT488.MAC, and link these programs with MSOFT.REL. The result is an executable file named MTSAMPL.COM. Since Pascal/MT+ uses the extension .ERL to denote relocatable (linkable) files, we rename MT488.REL and MSOFT.REL to MT488.ERL and MSOFT.ERL, respectively.

```
B>MTPLUS MTSAMPL<CR>
Pascal/MT+  Release 5.2
(c) 1980 MT MicroSYSTEMS

8080/Z80 Target CPU
++++++++++++++++++++++++
Source lines:   396
Available Memory: 12743
User Table Space:  8747
V5.2 Phase 1
############
Remaining Memory:  6942
V5.2 Phase 2
8080
INITVAR         39
ERRREPOR        99
GETCMD        1145
GETKEY        2317
PUTCHR        2362
APND          2392
CHARDEL       2446
GETSTR        2474
RESULTS       2790
GETPROTO      2937
PRNYN         3653
GETECHO       3722
SAMPLE
Lines :      396
Errors:        0
Code  :     5564
Data  :      472
Compilation Complete

B>M80 =MT488<CR>

No  Fatal error(s)

B>REN MT488.ERL=MT488.REL<CR>
B>REN MSOFT.ERL=MSOFT.REL<CR>
B>LINKMT MTSAMPL=MTSAMPL,MT488,MSOFT,PASLIB/S<CR>
Link/MT+ 5.2b


Processing file- MTSAMPL .ERL
Processing file- MT488   .ERL
Processing file- MSOFT   .ERL
Processing file- PASLIB  .ERL

Undefined Symbols:

No Undefined Symbols

0115 (decimal) records written to .COM file

Total Data: 06F5H bytes
Total Code: 3216H bytes
Remaining : 7165H bytes
```

```
Link/MT+ processing completed

B>
```

****** MTSAMPL.PAS Listing ******

```
program sample;

const
        SEQNUM = 0019;          (*editing sequence number*)
        TITLE = 0;              (*last edited ( 4/09/82-12:50)*)
        maxstr = 71;            (*maximum length of input string*)

(*      Let the operator test each function and observe the response

        Control characters (such as line feed and carriage return) can
        be entered into the TALK and CONTROL strings by preceding the
        control character with an ESCAPE.  For example, to get the string
        1234<ESCAPE>$$<RETURN><LINE FEED> you would type
        1234<ESCAPE><ESCAPE>$$<ESCAPE><RETURN><ESCAPE><LINE FEED><RETURN>
*)

var     er_code, time, poll, bus : integer;
        eot, eos, len : integer;
        echo_in, echo_out : integer;

        stop_flag : boolean;    (*determines if the user wants to abort*)

        cmd : integer;          (*holds number of command to execute*)

        bell : char;            (*holds ASCII BEL code*)
        bs : char;              (*holds ASCII back space code*)

        str : string[255];      (*string for general usage*)
        funct : string[20];     (*holds type of function for result report*)
        presp : string;         (*used for serial poll to return address of*)
                                (*  the device responding to the poll*)

        port : integer;         (*used when setting the P&T-488 port number*)

(*      The following are the declarations for the external procedures that
        are used to communicate to the 488 bus.  Note that not all of them
        are used by this program.                *)

external procedure pcntl (var s:string);
external procedure pcntlc (var s:string);
external procedure ptalk (var s:string);
external procedure ptalkc (var s:string);
external procedure pistn (var s:string);
external procedure pistnc (var s:string);
external procedure pspoll (var os,is : string);
external procedure pppoll;
external procedure pdren;
external procedure pren;
external procedure pstat;
external procedure pifc;
external procedure pbrset;
```

```
external procedure pioset (var ec,tv,pr,bs : integer);
external procedure pprot (var eot,eos,sl : integer);
external procedure pecho (var ei,eo : integer);
external procedure pioprt (var port:integer);

(*      The following external function allows direct access to BDOS functions*)
external function BBDOS (f:integer ; p:word) : integer;


(*-------------------------------------------------------------INITVAR-----*)
procedure initvar;
(*      Procedure to call the setup routines to tell MSOFT where the control
       variables are.          *)
begin
  pioset (er_code, time, poll, bus);
  pprot (eot, eos, len);
  pecho (echo_in, echo_out);
  bell:=chr(7);
  bs:=chr(8);
end;


(*-----------------------------------------------------------ERR_REPORT--*)
procedure err_report;
(*      Procedure to report the meaning of the error code.      *)
begin
  if er_code<>0 then
    if (er_code<0) or (er_code>255) then
      writeln('SYSTEM ERROR - an illegal error code has been encountered')
    else begin
      if tstbit(er_code,7) then
        writeln('S-100 RESET - reset interface (Use Function 6 or 7)');
      if tstbit(er_code,6) then
        writeln('IFC TRUE - reset 488 interface');
      if tstbit(er_code,5) then
        writeln('ATN TRUE - an external controller is trying to issue a command');
      if tstbit(er_code,4) then begin
        writeln('TIMEOUT ERROR - the specified amount of time has elapsed without');
        writeln('                completing a 488 handshake cycle');
        end;
      if tstbit(er_code,3) then
        writeln('SERVICE REQUEST - a 488 device is requesting service');
      if tstbit(er_code,2) then begin
        writeln('SERIAL POLL ADDRESS ERROR - no more than one secondary address');
        writeln('                            may follow a primary address');
        end;
      if tstbit(er_code,1) then
        writeln('NO LISTENERS - I cannot talk to myself');
      if tstbit(er_code,0) then begin
        writeln('SETUP ERROR - either IOSET or PROTCL wasn''t called before');
        writeln('              using one of the MSOFT communication functions');
        end;
      end;
end;


(*-------------------------------------------------------------GET_CMD-----*)
function get_cmd : integer;
```

```
(*      Function to present menu and input the code for the bus function
        to perform.             *)
var     i:integer;              (*variable for entry of function code*)
begin
  repeat
    writeln; writeln;
    writeln('1.  CONTROL  Become the Controller and output a command string');
    writeln('2.  TALK     Become a Talker and send a string');
    writeln('3.  LISTEN   Become a Listener and receive a string');
    writeln('4.  REMOTE   Make the REN (Remote ENable) line true');
    writeln('5.  LOCAL    Make the REN line false');
    writeln('6.  IFC      Issue an IFC (interface Clear) command');
    writeln('7.  RESET    Reset the P&T 488 interface');
    writeln('8.  STATUS   Display the current 488 bus status');
    writeln('9.  SPOLL    Perform a Serial Poll of the 488 bus');
    writeln('10. PPOLL    Perform a Parallel Poll of the 488 bus');
    writeln('11. Change the communication protocol (EOT switch, EOS, and string length)');
    writeln('12. Change input echo, output echo, and timeout values');
    writeln('13. Change S-100 port numbers (DIP switch on P&T-488 card must agree)');
    writeln('14. Exit to operating system');
    writeln;
    write('Which would you like to do? ');
    readln(i);
    if (i<1) or (i>14) then write(bell);
  until (i>0) and (i<15);
  get_cmd:=i;
end;


(*-------------------------------------------------------------------GET_KEY-----*)
function get_key : char;
(*      This function returns the next character from the console input.*)
var     ch : integer;
begin
  repeat
    ch:=@bdos(6,wrd(255));
  until ch<>0;
  get_key:=chr(ch);
end;


(*-------------------------------------------------------------------PUT_CHR-----*)
procedure put_chr (ch:char);
(*      This procedure puts a character out to the console using direct
        console I/O.    *)
var     dumy : integer;
begin
  dumy:=@bdos(6,wrd(ch));
end;


(*-------------------------------------------------------------------GET_STR-----*)
procedure get_str (var st : string);
(*      This procedure collects a string from the console with simple back
        space editing.  Control codes may be entered by preceding them by
        an escape.              *)
var     strlen : integer;               (*variable to track string length*)
        ch : char;                      (*variable for input character*)
```

```
(*.............................(GET_STR).....................APND.......*)
  procedure apnd;
  (*     This procedure is called to append a character onto the end of the
        string being collected.  It adjusts the string length and rings the
        bell if the string is already at its maximum length.    *)
  begin
    if strlen<maxstr then begin
      strlen:=succ(strlen);
      st[strlen]:=ch;
      end
    else
      put_chr(bell);
  end;

(*.............................(GET_STR).....................CHARDEL.....*)
  procedure chardel;
  (*     This procedure is called to delete a character.  It outputs <space>
        <back space><space>.    *)
  begin
    put_chr(bs); put_chr(' '); put_chr(bs);
  end;

begin                   (*GET_STR*)
  strlen:=0;                    (*set length to 0*)
  ch:=' ';                      (*init ch to a non-carriage return*)
  write('STRING: ');

  while ord(ch)<>13 do begin    (*collect until a carriage return*)
    ch:=get_key;                (*get a character from the console*)
    if (ch>=' ') and (ch<='~') then begin
      apnd;                     (*append character onto string*)
      put_chr(ch);              (*also echo it to the screen*)
      end
    else                        (*perform various control character fcns*)
      case ord(ch) of
        8:      if strlen>0 then begin          (*back space => delete char*)
                  if st[strlen]<' ' then chardel; (*need to delete 2 if ctl chr*)
                  chardel;
                  strlen:=pred(strlen);         (*adjust string length*)
                  end;

        27:     begin
                  ch:=get_key;                  (*get character after ESC to*)
                  apnd;                         (* put into string*)
                                        (*echo as printing char preceeded by @*)
                  put_chr('@'); put_chr(chr(ord(ch)+64));
                  ch:=' ';              (*make sure ch is not a carriage ret*)
                end;

        13:     ;

        else    put_chr(bell);          (*ring bell for invalid chars*)
      end;
    end;
  st[0]:=chr(strlen);                   (*set length byte of the returned str*)
end;
```

```
(*-------------------------------------------------------------RESULTS-----*)
procedure results;
(*       This procedure reports the results of a function.       *)
begin
  writeln;
  writeln('Function = ',funct,'        Error Code = ',er_code);
  if er_code=0 then writeln('NORMAL RETURN')
    else err_report;
end;


(*-----------------------------------------------------------GETPROTO----*)
procedure getproto;
(*       This procedure allows the user to set the EOT switch, EOS value,
        and the string length.  *)
begin
  writeln;writeln;
  writeln('The current communication protocol setup is:');
  writeln;
  writeln('    EOT switch      = ',eot);
  writeln('    EOS value       = ',eos);
  writeln('    String length   = ',len);
  writeln;
  write('What is the new EOT switch value? '); readln(eot);

  repeat
    write('What is the new EOS value (0..255)? '); readln(eos);
    if (eos<0) or (eos>255) then
      writeln('The EOS value must be between 0 and 255!');
  until (eos>=0) and (eos<=255);

  repeat
    write('What is the new String Length (0..255)? '); readln(len);
    if (len<0) or (len>255) then writeln('LENGTH must be between 0 and 255!');
  until (len>=0) and (len<=255);

  writeln;
end;


(*-------------------------------------------------------------GETECHO-----*)
procedure getecho;
(*       This procedure allows the user to respecify the input and output
        echo switches and the timeout.          *)
var     temp : string[10];

(*.............................(GETECHO).....................PRNYN.......*)
  procedure prnyn (v:integer);
  (*     This procedure prints 'N' if the passed parameter is 0 and 'Y'
        otherwise*)
  begin
    if v=0 then writeln('N') else writeln('Y');
  end;

begin                   (*GETECHO*)
  writeln; writeln;
  writeln('The Input Echo, Output Echo, and Timeout are currently set to:');
```

```
    writeln;
    write  ('    Input Echo      '); prnyn(echo_in);
    write  ('    Output Echo     '); prnyn(echo_out);
    writeln('    Timeout Value   ',time);
    writeln;

    repeat
      temp:=' ';
      write('Echo Input (Y/N) : '); readln(temp);
    until temp[1] in ['Y','y','N','n'];
    if temp[1] in ['Y','y'] then echo_in:=1 else echo_in:=0;

    repeat
      temp:=' ';
      write('Echo Output (Y/N) : '); read(temp);
    until temp[1] in ['Y','y','N','n'];
    if temp[1] in ['Y','y'] then echo_out:=1 else echo_out:=0;

    repeat
      write('What is the new TIMEOUT value (0..255)? '); readln(time);
      if (time<0) or (time>255) then
        writeln('The TIMEOUT value must be between 0 and 255!');
    until (time>=0) and (time<=255);
    writeln;
  end;


  begin   (*main program*)
    initvar;              (*initialize variables for control of MSOFT*)
    repeat
      cmd:=get_cmd;       (*get the function to perform*)
      er_code:=0;         (*clear error code*)
      case cmd of
        1:        begin
                    writeln('Please enter the Control string');
                    get_str(str);
                    funct:='CONTROLLER';
                    pcntl(str);
                  end;

        2:        begin
                    writeln('Please enter the Talk string');
                    get_str(str);
                    funct:='TALKER';
                    ptalk(str);
                  end;

        3:        begin
                    funct:='LISTENER';
                    plstn(str);
                    writeln('String heard on 488 bus is:');
                    writeln(str);
                    end;

        4:        begin
                    funct:='REMOTE ENABLE';
                    pren;
                  end;
```

```
          5:        begin
                      funct:='REMOTE DISABLE';
                      pdren;
                    end;

          6:        begin
                      funct:='INTERFACE CLEAR';
                      pifc;
                    end;

          7:        begin
                      funct:='RESET P&T 488';
                      pbrset;
                    end;

          8:        begin
                      funct:='STATUS';
                      pstat;
                      writeln('Bus Status is: ',bus);
                    end;

          9:        begin
                      funct:='Serial Poll';
                      writeln('Please enter Talk addresses to poll');
                      get_str(str);
                      writeln;
                      pspoll(str,presp);
                      writeln('Talk address of responding device is ',presp);
                      writeln('Poll response = ',poll);
                    end;

          10:       begin
                      funct:='Parallel Poll';
                      pppoll;
                      writeln('Poll response = ',poll);
                    end;

          11:       getproto;
          12:       getecho;
          13:       begin
                      write('What is the new S-100 port number (0-255)? ');
                      readln(port);
                      pioprt(port);
                    end;

          14:       stopflag:=true;
        end;

        if cmd<11 then results;

      until stopflag;

    end.
```

**** MT488.MAC ****

The program MT488.MAC performs all the parameter passing conversions necessary for a program written in Pascal MT+ to work with MSOFT.REL. Pascal MT+ passes parameters on the stack and expects the called routine to remove them from the stack before returning. MT488.MAC takes the parameters from the stack and puts them into the appropriate registers and tables for MSOFT.

```
            title   'Interface routines from Pascal/MT+ to MSOFT'

    SEQNUM  EQU     0009


    ;       The following are the entry points into MSOFT
            extrn   cnti,cntic,talk,talkc,istn
            extrn   istnc,spoll,ppoll,dren,ren
            extrn   status,ifc,brset,ioset,protcl
            extrn   echo,ioport

    ;       the following are the names used by MT+ programs to call the
    ;       MSOFT routines
            entry   pcnti, pcntic, ptalk, ptalkc, pistn
            entry   pistnc, pspoll, pppoll, pdren, pren
            entry   pstat, pifc, pbrset, pioset, pprot
            entry   pecho, pioprt

    ;-----------------------------------------------------------------------
    ;       General routine to call an MSOFT routine that has 1 string passed
    ;       to it
    ;       on entry:       DE => address of MSOFT routine to call
    ;-----------------------------------------------------------------------
    str1:   pop     b               ;get return address
            pop     h               ;get address of string
            mov     a,m             ;get length of string
            sta     dumy1
            inx     h               ;save address of string
            shld    dumy1a
            lxi     h,dumy1         ;get address of string pointer block
            push    b               ;put return address back on stack
            push    d               ;jump to target routine
            ret



    pcnti:  lxi     d,cnti          ;get address of MSOFT routine
            jmp     str1

    pcntic: lxi     d,cntic
            jmp     str1

    ptalk:  lxi     d,talk
            jmp     str1

    ptalkc: lxi     d,talkc
            jmp     str1

    pistn:  pop     h               ;get return address
            xthl                    ;swap it with address of MT+ string on stack
```

```
            shld    mtstr           ;save address where it won't be harmed

            lxi     h,dumy1         ;point to place for dope vector
            push    h               ;save address on stack
            call    lstn            ;call listen routine

lscomn:     pop     d               ;get dope vector address into DE
            ldax    d               ;get length of returned string
            lhld    mtstr           ;get address of MT+ string
            mov     m,a             ;set length of returned string
            ora     a               ;just return on 0 length
            rz
            mov     b,a             ;save length in b
            inx     d               ;point to addr field of dope vector
            ldax    d               ;get low byte of address
            mov     c,a             ;save it
            inx     d               ;high byte of addr field
            ldax    d               ;get it
            mov     d,a
            mov     e,c             ;DE holds address of MSOFT string
            inx     h               ;skip over count field of MT+ string

lslup:      ldax    d               ;transfer a character
            mov     m,a
            inx     h               ;increment pointers
            inx     d
            dcr     b               ;decrement count
            jnz     lslup           ;loop till done
            ret


pistnc:     pop     h               ;get return address
            xthl                    ;swap it with address of MT+ string on stack

            shld    mtstr           ;save address where it won't be harmed

            lxi     h,dumy1         ;point to place for dope vector
            push    h               ;save address on stack
            call    lstnc           ;call listen routine
            jmp     lscomn


pspoll:     pop     b               ;save return address
            pop     h               ;get address of string 2
            shld    mtstr           ;save it in a safe place

            pop     h               ;get address of string 1
            mov     a,m             ;set up dummy dope vector
            sta     dumy1
            inx     h
            shld    dumy1a
            push    b               ;restore return address
            lxi     h,dumy1
            lxi     d,dumy2
            push    d               ;put address of dummy dope vector on stack
            call    spoll           ;call serial poll routine
            jmp     lscomn          ;jump to routine to pass string back to MT+
```

```
pppoll: jmp     ppoll
pdren:  jmp     dren
pren:   jmp     ren
pstat:  jmp     status
pifc:   jmp     ifc
pbrset: jmp     brset

pioset: pop     b               ;save return address
        pop     h               ;get address of bus status variable
        shld    dumy4
        pop     h               ;get address of poll result variable
        shld    dumy3
        pop     d               ;get address of timeout value variable
        pop     h               ;get address of error code variable
        push    b               ;restore return address
        lxi     b,dumy3         ;point to additional parameters
        jmp     ioset           ;jump to MSOFT routine

pprot:  pop     h               ;save return address
        pop     b               ;get address of string length variable
        pop     d               ;get address of EOS variable
        xthl                    ;hl = address of EOT switch variable
                                ;tos = return address
        jmp     protcl          ;jump to MSOFT routine

pecho:  pop     b               ;save return address
        pop     d               ;get address of echoout variable
        pop     h               ;get address of echoin variable
        push    b               ;restore return address
        jmp     echo            ;jump   to MSOFT routine

pioprt: pop     b               ;save return address
        pop     h               ;get address of port variable
        push    b               ;restore return address
        jmp     ioport


dumy1:  db      0
dumy1a: dw      0

dumy2:  db      0
dumy2a: dw      0

dumy3:  dw      0
dumy4:  dw      0

mtstr:  dw      0

        end
```

****** MTCLOCK.PAS ******

This program performs the same function as the Basic program BICLOCK, but this one is written in Pascal MT+. Like BICLOCK, MTCLOCK initializes the 488 bus with an Interface Clear, puts the HP59309 clock into the Remote state by making the REN line true and sending the clock's listen address. It then addresses the clock as a Talker and listens to the data (status, date and time) that the clock sends over the bus. MTCLOCK displays the date and time on the console each time the minutes change. It also displays the data each time the status character indicates a clock error.

```
program mtclock;

const
        SEQNUM = 0012;          (*editing sequence number*)

(*      This is a Pascal/MT+ program which addresses an HP 59309A clock
        as a talker and then reads the time and date.  It continually
        rereads the time and displays the time and date on the console
        each minute.

        The program assumes that the bus output format of the 59309A is
        set to SPACE, CAL and COLON.  It also assumes that the talk address
        of the clock is "E" and the listen address is "%".      *)

var     er_code, time, poll, bus : integer;
        eot, eos, len : integer;
        echo_in, echo_out : integer;

        oldmin : char;          (*holds the previous value of minutes*)


(*      The following are the declarations for the external procedures that
        are used to communicate to the 488 bus.  Note that not all of them
        are used by this program.               *)

external procedure pcntl (var s:string);
external procedure pcntlc (var s:string);
external procedure ptalk (var s:string);
external procedure ptalkc (var s:string);
external procedure plstn (var s:string);
external procedure plstnc (var s:string);
external procedure pspoll (var os,is : string);
external procedure pppoll;
external procedure pdren;
external procedure pren;
external procedure pstat;
external procedure pifc;
external procedure pbrset;
external procedure ploset (var ec,tv,pr,bs : integer);
external procedure pprot (var eot,eos,si : integer);
external procedure pecho (var ei,eo : integer);

(*      The following external function allows direct access to BDOS functions*)
external function @BDOS (f:integer ; p:word) : integer;


(*-------------------------------------------------------INITVAR-----*)
procedure initvar;
```

```
(*       Procedure to call the setup routines to tell MSOFT where the control
        variables are.          *)
begin
  pioset (er_code, time, poll, bus);
  pprot (eot, eos, len);
  pecho (echo_in, echo_out);
end;


(*-------------------------------------------------------------ERR_REPORT--*)
procedure err_report;
(*       Procedure to report the meaning of the error code.      *)
begin
  if er_code<>0 then
    if (er_code<0) or (er_code>255) then
      writeln('SYSTEM ERROR - an illegal error code has been encountered')
    else begin
      if tstbit(er_code,7) then
        writeln('S-100 RESET - reset interface (use function IFC or BRSET)');
      if tstbit(er_code,6) then
        writeln('IFC TRUE - reset 488 interface');
      if tstbit(er_code,5) then
        writeln('ATN TRUE - an external controller is trying to issue a command');
      if tstbit(er_code,4) then begin
        writeln('TIMEOUT ERROR - the specified amount of time has elapsed without');
        writeln('                completing a 488 handshake cycle');
        end;
      if tstbit(er_code,3) then
        writeln('SERVICE REQUEST - a 488 device is requesting service');
      if tstbit(er_code,2) then begin
        writeln('SERIAL POLL ADDRESS ERROR - no more than one secondary address');
        writeln('                            may follow a primary address');
        end;
      if tstbit(er_code,1) then
        writeln('NO LISTENERS - I cannot talk to myself');
      if tstbit(er_code,0) then begin
        writeln('SETUP ERROR - either IOSET or PROTCL wasn''t called before');
        writeln('              using one of the MSOFT communication functions');
        end;
      end;
end;


(*-----------------------------------------------------------INITBUS-----*)
procedure initbus;
(*       Procedure to initialize the bus and set various control variables. *)
var     ctlstr : string[10];
begin
  pifc;                 (*do an interface clear*)
  pren;                 (*make the REN line true*)
  ctlstr:='? %';        (*Unlisten, Untalk, listen address %*)
  pcntlc(ctlstr);       (*become the controller and output CTLSTR*)
                        (* This puts the clock into the REMOTE mode*)
  if er_code<>0 then err_report;        (*report any bus errors*)
  time:=255;            (*do not time handshake*)
  eot:=1;               (*stop on End-Of-String byte*)
  eos:=10;              (*make line feed the EOS byte*)
  oldmin:='x';          (*set oldmin to some value which cannot match a clock*)
```

```
                        (*  reading*)
    end;


    (*-------------------------------------------------------------READ_TIME---*)
    function read_time : boolean;
    (*      Function to read the clock and display results on the console.
           Does not return until either the user aborts operation or a bus error
           occurs.  Returns true if the user aborts or a clock error occurs.
           Returns false if a bus error occurred.  *)
    var    reading : string[128];          (*string to read clock into*)
           check : integer;                (*used for abort checking*)
           ctlstr : string[10];            (*used to send control string*)
    begin
      repeat
        ctlstr:='?_E';      (*set control string for Unlisten, Untalk, Talk addr E*)
        pcntlc(ctlstr);     (*send control string*)
        if er_code<>0 then err_report       (*report any errors*)
        else begin
          pistnc(reading);                  (*read the clock*)
          if er_code<>0 then err_report     (*report any errors*)
          else begin
                   (*if the first character is a ? then the clock is in error*)
            if reading[1]='?' then begin
              writeln('Clock error  ',reading);
              writeln('Reset clock');
              end
            else            (*show the time if the minutes have changed*)
              if reading[13]<>oldmin then writeln(reading);
            oldmin:=reading[13];
            end;
          check:=@bdos(6,wrd(255));         (*check for character at keyboard*)
          end;
      until (er_code<>0) or (reading[1]='?') or (check=3);
      read_time:=(check=3) or (reading[1]='?');     (*set returned value*)
    end;


    begin   (*main program*)
      initvar;              (*initialize variables for control of MSOFT*)
      repeat
        initbus;                    (*initialize bus*)
      until read_time or (@bdos(6,wrd(255))=3);
    end.
```

****** FSAMPL.FOR ******

This program performs the same function as the Basic program BISAMPL, but this one is written in Microsoft Fortran.

The following dialog shows how to compile the Fortran program FSAMPL.FOR, assemble the assembler program STRIN.MAC, and then link these two programs with MSOFT.REL. The result is an executable file named CLOCK.COM.

```
B>F80 =FSAMPL<CR>
$MAIN
ERRMSG
STRXFR
IUNSGN

B>M80 =STRIN<CR>

No  Fatal error(s)

B>L80 FSAMPL/E,FSAMPL/N,MSOFT,STRIN<CR>

Link 80   3.42  19-Feb-81  Copyright (c) 1981 Microsoft

Data      0103   3DAD    <15530>

23862 Bytes Free
1098F   3DAD    611

B>
```

```
C    FORTRAN driver for MSOFT
C    FSAMPL.FOR   revised 4/8/82 by J. Tinsman
C
C    NOTE:  The name that MBASIC uses for the echo out variable
C           is ECHOOUT, but the longest variable name FORTRAN will
C           accept is ECHOUT (6 characters).  This program uses the
C           substitute name ECHOUT.
C
C   Let the operator test each function and observe the response
C
C Control characters (such as line feed and carriage return) can
C be entered into the TALK and CONTROL strings by preceeding the
C control character with an ESCAPE.  For example, to get the string
C 1234<ESCAPE>$$<RETURN><LINE FEED> you would type
C 1234<ESCAPE><ESCAPE>$$<ESCAPE><RETURN><ESCAPE><LINE FEED><RETURN>.
C
C
C
      INTEGER ERCODE,TIME,EOT,EOS,LENGTH,POLL,ECHOIN,ECHOUT,BUS
      INTEGER I,J,P,F,BELL,BUFLEN
      BYTE BUFFER(255),TKADDR(5)
      DOUBLE PRECISION FCL,FCH,FCNS(2,12)
C
     DATA FCNS /'CONTROLL','ER','TALKER',' ','LISTENER',' ','REMOTE E',
     _'NABLE','REMOTE D','ISABLE','INTERFAC','E CLEAR','RESET P&','T',
     _'STATUS',' ','SERIAL P','OLL','PARALLEL',' POLL',' ',' ',' ',' '/
C
```

```
      DATA ERCODE,TIME,EOT,EOS,LENGTH,POLL,ECHOIN,ECHOUT,BUS /9*0/
C
       BELL=7
       BUFLEN=255
C
C  Pass variable names that FSAMPL will be using to MSOFT
C
       CALL IOSET (ERCODE,TIME,POLL,BUS)
       CALL PROTCL (EOT,EOS,LENGTH)
       CALL ECHO (ECHOIN,ECHOUT)
C
C
C   ================================================
C   Main Menu
C
 530   WRITE (1,531)
 531   FORMAT ('0')
       WRITE (1,541)
 541   FORMAT(' 1.  CONTROL   Become the Controller and output a command
     _ string')
       WRITE (1,551)
 551   FORMAT (' 2.  TALK      Become a Talker and send a string')
       WRITE (1,561)
 561   FORMAT (' 3.  LISTEN    Become a Listener and receive a string')
       WRITE (1,571)
 571   FORMAT (' 4.  REMOTE    Make the REN (Remote ENable) line true')
       WRITE (1,581)
 581   FORMAT (' 5.  LOCAL     Make the REN line false')
       WRITE (1,591)
 591   FORMAT (' 6.  IFC       Issue an IFC (InterFace Clear) command')
       WRITE (1,601)
 601   FORMAT (' 7.  RESET     Reset the P&T 488 interface')
       WRITE (1,611)
 611   FORMAT (' 8.  STATUS    Display the current 488 bus status')
       WRITE (1,621)
 621   FORMAT (' 9.  SPOLL     Perform a Serial Poll of the 488 bus')
       WRITE (1,631)
 631   FORMAT (' 10. PPOLL     Perform a Parallel Poll of the 488 bus')
       WRITE (1,641)
 641   FORMAT (' 11. Change the communication protocol (EOT switch, EOS,
     _and string length)')
       WRITE (1,651)
 651   FORMAT (' 12. Change input echo, output echo and timeout values')
       WRITE (1,671)
 671   FORMAT ( '0','Which would you like to do? ')
       READ (3,673) F
 673   FORMAT (I5)
 680   IF (F.GT.0.AND.F.LT.13) GOTO 690
       WRITE (1,681) BELL
 681   FORMAT (' ',A1)
       GOTO 530
 690   WRITE (1,691)
 691   FORMAT (' ')
C
       ERCODE=0
       FCH=FCNS(1,F)
       FCL=FCNS(2,F)
C
```

```
C
 720   IF (F.NE.1) GOTO 790
       WRITE (1,731)
 731   FORMAT (' Please enter the Control string')
C   Get string to send as a controller
       CALL STRIN (BUFFER,BUFLEN)
C   Send out the command string
       CALL CNTL (BUFFER)
       GOTO 1790
C
C
 790   IF (F.NE.2) GOTO 860
       WRITE (1,801)
 801   FORMAT (' Please enter the Talk string')
C   Get string to send as a talker
       CALL STRIN (BUFFER,BUFLEN)
C   Send out the talk string
       CALL TALK (BUFFER)
       GOTO 1790
C
C
 860   IF (F.NE.3) GOTO 940
C
C   Read string off bus into P&T 488 buffer and make string descriptor
C   in the first three elements of array BUFFER
       CALL LSTN (BUFFER)
C   Transfer contents of P&T 488 buffer into array BUFFER
       CALL STRXFR (BUFFER,BUFLEN)
       WRITE (1,901)
 901   FORMAT (' String heard on 488 bus is: ')
       J=IUNSGN(BUFFER(1))+3
       WRITE (1,911) (BUFFER(I),I=4,J)
 911   FORMAT (' ',255A1)
       GOTO 1790
C
C
 940   IF (F.NE.4) GOTO 990
C
C   Make REN line true
       CALL REN
       GOTO 1790
C
C
 990   IF (F.NE.5) GOTO 1040
C
C   Make REN line false
       CALL DREN
       GOTO 1790
C
C
 1040  IF (F.NE.6) GOTO 1090
C
C   Issue an IFC command
       CALL IFC
       GOTO 1790
C
C
 1090  IF (F.NE.7) GOTO 1140
```

```
C
C   Reset the P&T 488
      CALL BRSET
      GOTO 1790
C
C
 1140 IF (F.NE.8) GOTO 1200
      CALL STATUS
      WRITE (1,1161) BUS
 1161 FORMAT (' Bus status is: ',I3)
      GOTO 1790
C
C
 1200 IF (F.NE.9) GOTO 1300
      WRITE (1,1211)
 1211 FORMAT (' Please enter Talk address to poll')
C
C   Get talk address string
      CALL STRIN (BUFFER,BUFLEN)
      WRITE (1,1231)
 1231 FORMAT (' ')
C
C   Send out Talk string and put response in P&T 488 string buffer.
C   Then put string descriptor for response in the first three
C   elements of TKADDR.
      CALL SPOLL (BUFFER,TKADDR(1))
      IF (TKADDR(3).NE.0) GOTO 1260
      I=' '
      TKADDR(4)=I
      TKADDR(5)=I
C   Transfer response from P&T 488 buffer to array TKADDR
 1260 CALL STRXFR (TKADDR,5)
      WRITE (1,1261) TKADDR(4),TKADDR(5)
 1261 FORMAT (' Talk address of device responding is ',2A1)
      WRITE (1,1271) POLL
 1271 FORMAT (' Poll response = ',I3)
      GOTO 1790
C
C
 1300 IF (F.NE.10) GOTO 1360
C
C   Perform parallel poll
      CALL PPOLL
      WRITE (1,1331) POLL
 1331 FORMAT (' Poll response = ',I3)
      GOTO 1790
C
C
 1360 IF (F.NE.11) GOTO 1560
      WRITE (1,1371)
 1371 FORMAT (' ')
      WRITE (1,1381)
 1381 FORMAT ('0','The current communication protocol setup is:')
      WRITE (1,1371)
      WRITE (1,1401) EOT
 1401 FORMAT (' ',10X,'EOT switch       = ',I3)
      WRITE (1,1411) EOS
 1411 FORMAT (' ',10X,'EOS switch       = ',I3)
```

```
      WRITE (1,1421) LENGTH
 1421 FORMAT (' ',10X,'String length     = ',I3)
      WRITE (1,1441)
 1441 FORMAT ('0', 'What is the new EOT switch value: ')
      READ (3,1445) EOT
 1445 FORMAT (I4)
 1450 WRITE (1,1451)
 1451 FORMAT (' What is the new EOS value: ')
      READ (3,1445) EOS
      IF (EOS.GE.0.AND.EOS.LT.256) GOTO 1490
      WRITE (1,1471)
 1471 FORMAT (' The EOS value must be between 0 and 255!!!')
      GOTO 1450
 1490 WRITE (1,1491)
 1491 FORMAT (' What is the new string length: ')
      READ (3,1445) LENGTH
      IF (LENGTH.GE.0.AND.LENGTH.LT.256) GOTO 1530
      WRITE (1,1511)
 1511 FORMAT (' The LENGTH must be between 0 and 255!!!')
      GOTO 1490
 1530 WRITE (1,1531)
 1531 FORMAT (' ')
      GOTO 530
C
C
 1560 IF (F.NE.12) GOTO 1790
      WRITE (1,1571)
 1571 FORMAT ('0')
 1581 FORMAT (' The Input Echo, Output Echo, and Timeout values are cu
     _rrently set to: ')
      P='N'
      IF (ECHOIN.NE.0) P='Y'
      WRITE (1,1611) P
 1611 FORMAT (' ',10X,'Input Echo          ',A1)
      P='N'
      IF (ECHOUT.NE.0) P='Y'
      WRITE (1,1621) P
 1621 FORMAT (' ',10X,'Output Echo         ',A1)
      WRITE (1,1641) TIME
 1641 FORMAT (' ',10X,'Timeout Value       ',I3)
 1660 WRITE (1,1661)
 1661 FORMAT ('0','Echo Input (Y/N) : ')
      READ (3,1671) P
 1671 FORMAT (A1)
      IF (P.NE.'Y'.AND.P.NE.'N'.AND.P.NE.8313.AND.P.NE.8302) GOTO 1660
      ECHOIN=0
      IF (P.EQ.'Y'.OR.P.EQ.8313) ECHOIN=1
 1710 WRITE (1,1711)
 1711 FORMAT ('0','Echo Output (Y/N) : ')
      READ (3,1721) P
 1721 FORMAT (A1)
      IF (P.NE.'Y'.AND.P.NE.'N'.AND.P.NE.8313.AND.P.NE.8302) GOTO 1710
      ECHOUT=0
      IF (P.EQ.'Y'.OR.P.EQ.8313) ECHOUT=1
 1750 WRITE (1,1761)
 1761 FORMAT (' What is the new TIMEOUT value: ')
      READ (3,1771) TIME
 1771 FORMAT (I4)
```

```
      IF (TIME.GE.0.AND.TIME.LT.256) GOTO 530
      WRITE (1,1781)
 1781 FORMAT (' The TIMEOUT value must be between 0 and 255!!!')
      GOTO 1750
C
C
 1790 WRITE (1,1791)
 1791 FORMAT (' ')
      CALL ERRMSG(ERCODE,FCH,FCL)
      GOTO 530
      END
C
C

      SUBROUTINE ERRMSG (ERCODE,FCH,FCL)
C
       INTEGER ERCODE,I,J,K,IO,R9
       DOUBLE PRECISION FCH,FCL
C
C                          Report 488 Function Errors
C
      WRITE (1,9031) FCH,FCL,ERCODE
 9031 FORMAT (' Function = ',A8,A8,14X,'Error Code = ',I3)
C
C Interpret Error codes and print error messages
C
      IF (ERCODE.LT.0) GOTO 9370
      IF (ERCODE.NE.0) GOTO 9090
          WRITE (1,9085)
 9085      FORMAT (' NORMAL RETURN')
          RETURN
 9090 IF (ERCODE.GT.255) GOTO 9370
      DO 9350 K=0,7
      I=7-K
      IO=2**I
      R9=ERCODE-IO
      IF (R9.LT.0) GOTO 9350
      ERCODE=R9
      J=I+1
      GOTO (9160,9190,9210,9240,9260,9290,9310,9330),J
C
 9160 WRITE (1,9161)
 9161  FORMAT (' SETUP ERROR - either IOSET or PROTCL was not called be
     _fore')
C
 9170 WRITE (1,9171)
 9171  FORMAT ('               using one of the MSOFT communication fun
     _ctions')
      GOTO 9350
C
 9190 WRITE (1,9191)
 9191  FORMAT (' NO LISTENERS - I cannot talk to myself!')
      GOTO 9350
C
 9210 WRITE (1,9211)
 9211  FORMAT (' SERIAL POLL ADDRESS ERROR - no more than one secondary
     _address')
 9220 WRITE (1,9221)
```

```
 9221   FORMAT ('                                  may follow a primary addre
     _ss')
        GOTO 9350
C
 9240  WRITE (1,9241)
 9241   FORMAT (' SERVICE REQUEST - a 488 device is requesting service')
        GOTO 9350
C
 9260  WRITE (1,9261)
 9261   FORMAT (' TIMEOUT ERROR - the specified amount of time has elaps
     _ed without')
 9270  WRITE (1,9271)
 9271   FORMAT ('                  completing a 488 handshake cycle')
        GOTO 9350
C
 9290  WRITE (1,9291)
 9291   FORMAT (' ATN TRUE - an external controller is trying to issue a
     _command')
        GOTO 9350
C
 9310  WRITE (1,9311)
 9311   FORMAT (' IFC TRUE - reset 488 interface')
        GOTO 9350
C
 9330  WRITE (1,9331)
 9331   FORMAT (' S-100 RESET - reset interface (use function I or R)')
        GOTO 9350
C
 9350  CONTINUE
 9360  RETURN
C
 9370  WRITE (1,9371)
 9371   FORMAT (' SYSTEM ERROR - an illegal error code has been encounte
     _red')
 9380  RETURN
       END
C
C
C   STRING TRANSFER ROUTINE
C
       SUBROUTINE STRXFR (ARRAY,SIZE)
C
       INTEGER I,J,SIZE,STRLEN,ADDR
       BYTE ARRAY
       DIMENSION ARRAY(SIZE)
C
       STRLEN=IUNSGN(ARRAY(1))
       ADDR=256*IUNSGN(ARRAY(3))+IUNSGN(ARRAY(2))
C
       J=STRLEN+3
       IF (SIZE.GE.J) GOTO 9600
       WRITE (1,9505)
 9505  FORMAT (' THE STRING RECEIVED IS BIGGER THAN THE ARRAY GIVEN!!')
       WRITE (1,9506) SIZE
 9506  FORMAT (' Only the first ',I3,' characters were transferred.')
C
       DO 9520 I=4,SIZE
          ARRAY(I)=PEEK(ADDR)
```

```
            ADDR=ADDR+1
 9520   CONTINUE
        RETURN
C
C
 9600   DO 9610 I=4,J
            ARRAY(I)=PEEK(ADDR)
            ADDR=ADDR+1
 9610   CONTINUE
C
        J=J+1
        DO 9620 I=J,SIZE
            ARRAY(I)=0
 9620   CONTINUE
        RETURN
        END
C
C
C   FUNCTION IUNSGN
C
        INTEGER FUNCTION IUNSGN(M)
C
        BYTE M
        INTEGER IUNSGN
C
        IUNSGN=M
        IF (IUNSGN.LT.0) IUNSGN=IUNSGN+256
        RETURN
        END
```

****** STRIN.MAC ******

This assembly program is used by FSAMPL.FOR to collect strings from the keyboard. It is the routine responsible for the capability of entering a control character (such as line feed or carriage return) in the string by preceding the control character with an ESCAPE. It is also responsible for displaying the string on the console, and allowing the operator to delete characters by pressing backspace.

This routine was written in assembly language instead of Fortran because Fortran does not have string manipulation capability. This program could be written in Fortran but the code would be even more difficult to understand.

```
;               STRIN (bufname,buflen)       4/8/82
;
;
; STRING INPUT ROUTINE FOR USE WITH MICROSOFT FORTRAN
; THIS ROUTINE EXPECTS TWO PARAMETERS TO BE PASSED TO IT:
;       THE NAME OF THE BUFFER THE STRING IS TO BE PUT IN
;       AND THE LENGTH OF THE BUFFER (UP TO 255)
;
; THE BUFFER ARRAY VARIABLE MUST BE OF BYTE TYPE.  THE BUFFER LENGTH
; VARIABLE MUST BE OF INTEGER TYPE.  THIS ROUTINE ONLY LOOKS AT THE
; LOW ORDER BYTE OF THE INTEGER VARIABLE.
;
; NOTE THAT THE ACTUAL USABLE BUFFER SIZE IS THE LENGTH PASSED-3. THIS IS
; BECAUSE THE FIRST THREE BYTES OF THE BUFFER ARE USED TO CREATE THE STRING
```

```
; DESCRIPTOR THAT MSOFT NEEDS TO SEND OUT STRINGS.  THE FIRST BYTE OF THE
; DESCRIPTOR GIVES THE STRING LENGTH; THE SECOND AND THIRD BYTES GIVE THE
; STRING'S STARTING ADDRESS.
;
        PUBLIC STRIN
;
BDOS    EQU     0005        ;BDOS JUMP VECTOR
DRCTIO  EQU     06H         ;BDOS FUNCTION FOR UNBUFFERED I/O
VERNUM  EQU     0CH         ;BDOS FUNCTION FOR CP/M VERSION NUMBER
STROUT  EQU     09H         ;BDOS FUNCTION FOR STRING OUTPUT
  ;
BELL    EQU     07H         ;BELL
BS      EQU     08H         ;BACKSPACE
CR      EQU     0DH         ;CARRIAGE RETURN
CARET   EQU     05EH        ;CARET
ESC     EQU     01BH        ;ESCAPE
LF      EQU     0AH         ;LINE FEED
SPACE   EQU     020H        ;SPACE
;
;
STRIN:  MVI     M,00        ;INIT STRING LENGTH TO ZERO AND SAVE IN BUFFER
        SHLD    LENPT       ;SAVE POINTER TO LENGTH COUNT IN LENPT
        INX     H           ;POINT HL TO WHERE STRING START ADDRESS WILL GO
        MOV     B,H         ;COPY HL INTO BC
        MOV     C,L         ;
        INX     B           ;INCREMENT IT TWO TIMES TO GET
        INX     B           ;ACTUAL STRING STARTING ADDRESS
        MOV     M,C         ;SAVE STRING STARTING ADDRESS
        INX     H           ;INTO STRING DESCRIPTOR PART OF BUFFER
        MOV     M,B         ;
        INX     H           ;POINT HL TO START OF STRING IN BUFFER
        SHLD    BUFPT       ;SAVE STRING POINTER
        XCHG
        MOV     A,M         ;GET MAX BUFFER LENGTH FROM LOW ORDER BYTE
        STA     BUFMAX      ;OF BUFLEN VARIABLE AND SAVE IT
;
        MVI     C,VERNUM    ;CHECK FOR CP/M VERSION NUMBER
        CALL    BDOS
        MOV     A,H         ;RESULT IS RETURNED IN H,L
        ORA     L           ;IF EITHER H OR L <>0, THEN THE
        JNZ     STRIN1      ;CP/M IS VERSION 2.0 OR LATER
        MVI     C,STROUT    ;IF BOTH ARE ZERO, PRINT ERROR MESSAGE
        LXI     D,VERERR
        JMP     BDOS
;
STRIN1: MVI     C,STROUT    ;PRINT PROMPT
        LXI     D,PROMPT
        CALL    BDOS
        SUB     A
        STA     BUFCT       ;INIT BUFFER COUNT
;
STRIN2: CALL    GETCHR      ;GET A CHARACTER FROM CONSOLE
        CPI     CR          ;CHECK FOR CARRIAGE RETURN
        JNZ     STRIN3
        LDA     BUFCT       ;GET STRING LENGTH
        LHLD    LENPT       ;GET POINTER TO LENGTH DEST. IN BUFFER
        MOV     M,A         ;SAVE STRING LENGTH IN BUFFER
        LDA     CR          ;SEND OUT CARRIAGE RETURN
```

```
            JMP     CHROUT
;
STRIN3:     CPI     BS          ;CHECK FOR BACKSPACE
            JZ      BAKSPC
            CPI     ESC         ;CHECK FOR ESCAPE KEY
            JNZ     STRIN4      ;IF NOT, THEN SKIP OVER ESCAPE HANDLER
            CALL    GETCHR      ;GET KEY FOLLOWING ESC
            JMP     STRIN5      ;SKIP CONTROL CHARACTER CHECK
STRIN4:     CPI     SPACE       ;CHECK FOR CONTROL CHARACTER
            JC      STRIN2      ;IF SO, IGNORE
;
STRIN5:     STA     KEYBUF      ;SAVE KEY
            LDA     BUFMAX      ;GET MAX BUFFER SIZE
            SUI     03          ;SUBTRACT THREE TO GET TRUE BUFFER USAGE
            MOV     B,A
            LDA     BUFCT       ;GET CURRENT BUFFER USAGE
            CMP     B           ;IF TWO ARE THE SAME, THEN BUFFER IS FULL
            JZ      BUFFUL      ;JUMP TO BUFFER ERROR HANDLER
            INR     A           ;INCREMENT COUNT
            STA     BUFCT       ;SAVE IT
            LHLD    BUFPT       ;GET BUFFER POINTER
            LDA     KEYBUF      ;GET KEY BACK
            MOV     M,A         ;SAVE CHARACTER IN BUFFER
            INX     H           ;INCREMENT BUFFER POINTER
            SHLD    BUFPT       ;SAVE IT
            CALL    PUTCHR      ;ECHO CHARACTER TO CONSOLE
            JMP     STRIN2      ;LOOP BACK
;
BUFFUL:     MVI     A,BELL      ;SET UP TO RING BELL
            CALL    CHROUT      ;SEND IT
            JMP     STRIN2      ;LOOP BACK (IGNORING LAST CHARACTER TYPED)
;
GETCHR:     MVI     C,DRCTIO    ;SET UP FOR KEY FETCH
            MVI     E,0FFH
            CALL    BDOS        ;GET KEY
            ANI     07FH        ;STRIP OFF PARITY
            JZ      GETCHR      ;IF RESULT ZERO, NO KEY WAS PRESSED
            RET
;
PUTCHR:     STA     PUTBUF      ;SAVE CHARACTER TO BE PRINTED
            CPI     SPACE       ;COMPARE TO SPACE
            JNC     CHROUT      ;IF NO CARRY, PRINT CHARACTER
            MVI     A,CARET     ;OTHERWISE, IT'S A CONTROL CHARACTER
            CALL    CHROUT      ;SO PRINT CARET FIRST
            LDA     PUTBUF      ;GET SAVED CHARACTER
            ADI     040H        ;ADD 64 TO CONVERT TO PRINTING CHARCTER
CHROUT:     MVI     C,DRCTIO    ;SET UP FOR SINGLE CHARACTER PRINT
            MOV     E,A         ;MOVE CHARACTER TO BE SENT INTO E
            JMP     BDOS        ;SEND CHARACTER
;
BAKSPC:     LDA     BUFCT       ;GET STRING LENGTH
            CPI     00
            JZ      STRIN2      ;IF LENGTH IS ZERO, THEN NO BACKSPACE
            DCR     A           ;DECREMENT LENGTH
            STA     BUFCT       ;SAVE IT
            CALL    BACKUP      ;DELETE CHARACTER FROM CONSOLE
            LHLD    BUFPT       ;GET BUFFER POINTER
            DCX     H           ;DECREMENT IT
```

```
        SHLD    BUFPT       ;SAVE IT
        MOV     A,M         ;GET LAST CHARACTER
        CPI     SPACE       ;CHECK FOR CONTROL CHARACTER
        CC      BACKUP      ;IF SO, ERASE PRECEDING CARET FROM CONSOLE
        JMP     STRIN2      ;LOOP BACK
;
BACKUP: MVI     A,BS        ;SEND OUT BACKSPACE ,SPACE,BACKSPACE
        CALL    CHROUT
        MVI     A,SPACE
        CALL    CHROUT
        MVI     A,BS
        JMP     CHROUT
;
;
BUFCT:  DS      1           ;CURRENT STRING LENGTH
BUFPT:  DS      2           ;POINTER TO NEXT AVAILABLE BUFFER LOCATION
BUFMAX: DS      1           ;BUFFER LENGTH
KEYBUF: DS      1           ;ONE CHARACTER KEY BUFFER
LENPT:  DS      2           ;POINTER TO LENGTH LOCATION IN BUFFER
PUTBUF: DS      1
PROMPT: DB      CR,LF,'STRING:$'
VERERR: DB      CR,'STRING INPUT ROUTINE REQUIRES CP/M VERSION 2.0 '
        DB      'OR LATER!',CR,'$'
;
        END
```

**** FCLOCK.FOR ****

Like all the other clock programs, this one performs the same function but is written in Microsoft Fortran. It initializes the 488 bus by sending an Interface Clear (IFC), puts the clock into the remote mode by making the REN line true and then addressing the clock as a listener. It then addresses the clock as a talker and listens to the data (status, date and time) that the clock sends over the bus. It displays the date and time each time the minutes change. It also displays the data each time the status character indicates a clock error.

A few differences will be seen when FCLOCK is compared to BICLOCK. Fortran does not have much ability to manipulate strings, so a special array is created to hold the data string read from the clock. This array is called BUFFER. The first three bytes of the array are used to emulate Basic's string descriptor block. Remember that the first byte of the string descriptor block holds the length of the string, and the remaining two bytes hold the address of the string. Similar arrays are set up for the two strings TAD and LAD.

A routine that you will find useful if you write Fortran programs for MSOFT is STRXFR. It transfers (copies) strings from MSOFT's input string buffer into the specified array. Another useful routine is STRSET, which generates a string descriptor block in the first three bytes of an array.

FCLOCK.FOR is compiled and linked in just the same way that FSAMPL.FOR is. If you follow the dialog shown for FSAMPL, subsituting FCLOCK each place FSAMPL appears, and IVARPT each place STRIN appears, you will get an execuatble file named FCLOCK.COM.

```
C  ================================================
C
C                        FCLOCK.FOR
C
C       Revised for MSOFT.REL by John Tinsman  4-15-82
C
C    This is a Microsoft FORTRAN program which addresses an
C  an HP 59309A clock as a talker and then reads the time and
C  date.  It continually rereads the time and displays the
C  time and date on the console each minute.
C
C  The program assumes that the bus output format of the 59309A
C  is set to SPACE, CAL, and COLON.  It also assumes that the
C  TALK address of the clock is "E" and the listen address is "%".
C
C  ================================================
C
      INTEGER ERCODE, TIME, EOT, EOS, LENGTH, POLL, ECHOIN, ECHOUT, BUS
      INTEGER STATUS, MIN, OLDMIN
      BYTE BUFFER(23),TAD(6),LAD(6)
C
C       The byte array TAD contains an Unlisten and an Untalk command
C   followed by the clock's talk address in the last three bytes.  The
C   first three bytes are used to store the string descriptor for the
C   last three bytes.  The byte array LAD is almost the same, but the
C   last byte of the string is -- instead of being the clock's talk
C   address -- the clock's listen address.  In both cases, the first 3
C   bytes (which form the string descriptor) are initially set to 0,
C   and then later set to the proper values by using the subroutine
C   STRSET to do the string descriptor set ups.
C
      DATA TAD /0,0,0,'?','_','E'/
```

```
      DATA LAD /0,0,0,'?','_','$'/
C
C  Pass variable names that FCLOCK will be using to MSOFT
C
      CALL IOSET (ERCODE,TIME,POLL,BUS)
      CALL PROTCL (EOT,EOS,LENGTH)
      CALL ECHO (ECHOIN,ECHOUT)
C
C  Intialize OLDMIN to some value which cannot match the first reading
C  from the clock.  This will insure that the time and date will be
C  displayed the first time through.
C
      OLDMIN = -1
C
C  Issue an IFC command
C
 30   CALL IFC
C
C  Make REN line true
C
      CALL REN
C
C  TIME contains the amount of time to allow for handshake
C  If TIME=255, then the handshake is not timed
C
      TIME = 255
C
C  Turn off input and output echo
C
      ECHOIN = 0
      ECHOUT = 0
C
C  Set up the Listen string.  It contains the string descriptor (3 bytes),
C  the UNLISTEN byte, the UNTALK byte and the Listen Address of the clock.
C
      CALL STRSET(LAD,6,3)
C
C  Become the 488 controller and issue UNLISTEN, UNTALK and then
C  address the clock as a listener.
C  This puts the clock in the REMOTE mode.
C
      CALL CNTLC(LAD)
      IF (ERCODE.NE.0) GOTO 200
C
C
C  Set up MSOFT so it will stop on EOS (End-of-String) byte, set the EOS
C  to be a line feed.
C
      EOT = 1
      EOS = 10
C
C  Set up the Talk string.  It contains the string descriptor (3 bytes),
C  the UNLISTEN byte, the UNTALK byte and the Talk Address of the clock.
C
      CALL STRSET(TAD,6,3)
C
C  Become the 488 controller and issue UNLISTEN, UNTALK and then
C  address the clock as a talker.
```

```
C
 40   CALL CNTLC (TAD)
C
C  Report any 488 errors that may have occurred
C
      IF (ERCODE.NE.0) GOTO 200
C
C  Become a listener and read the time from the clock
C
 50   CALL LSTNC(BUFFER)
C
C  Report any 488 errors that may have occurred
C
      IF (ERCODE.NE.0) GOTO 200
C
C  Transfer clock response from 488 buffer into array BUFFER
C
      CALL STRXFR(BUFFER,23)
C
C  The clock's response is now stored in elements 4-23 of the byte array
C  BUFFER in the following form:
C
C    ¶? or <sp>¶<sp>¶<:>¶M¶M¶<:>¶D¶D¶<:>¶H¶H¶<:>¶M¶M¶<:>¶S¶S¶<cr>¶<lf>¶
C
C   Put BUFFER(4) into STATUS, and put the string length + 3 into J
C
      STATUS = BUFFER(4)
      J = IUNSGN(BUFFER(1))+3
C
C   Put the least significant digit of the minutes into MIN
C
      MIN = BUFFER(16)
C
C   Check the clock status.  If it's not OK then print message and halt
C
      IF (STATUS.EQ.32) GOTO 100
      WRITE (1,55) (BUFFER(I),I=4,J)
      WRITE (1,60)
 55   FORMAT (' ',20A1)
 60   FORMAT (' Reset clock')
      WRITE (1,55)
      STOP
C

C   Show the time if the minutes have changed
C
 100  IF (MIN.NE.OLDMIN) WRITE (1,110) (BUFFER(I),I=4,J)
 110     FORMAT (' ',20A1)
C
C   Update OLDMIN and read clock again
C
      OLDMIN = MIN
      GOTO 40
C
C   Error handling routine
C   If an error occurs, print error message and go back to
C   IFC, REN, etc.
C
```

```
  200  CALL ERRMSG(ERCODE)
       GOTO 30
C
       END
C  ------------------------------------------ ERRMSG --------
C  488 Interface error reporting routine
C
       SUBROUTINE ERRMSG (ERCODE)
C
       INTEGER ERCODE,I,J,K,I0,R9
C
       WRITE (1,9031) ERCODE
 9031  FORMAT (' Error Code = ',I3)
C
C Interpret Error codes and print error messages
C
       IF (ERCODE.LT.0) GOTO 9370
       IF (ERCODE.NE.0) GOTO 9090
          RETURN
 9090  IF (ERCODE.GT.255) GOTO 9370
       DO 9350 K=0,7
       I=7-K
       I0=2**I
       R9=ERCODE-I0
       IF (R9.LT.0) GOTO 9350
       ERCODE=R9
       J=I+1
       GOTO (9160,9190,9210,9240,9260,9290,9310,9330),J
C
 9160  WRITE (1,9161)
 9161   FORMAT (' SETUP ERROR - either IOSET or PROTCL was not called be
     _fore')
C
 9170  WRITE (1,9171)
 9171   FORMAT ('                  using one of the MSOFT communication fun
     _ctions')
       GOTO 9350
C
 9190  WRITE (1,9191)
 9191   FORMAT (' NO LISTENERS - I cannot talk to myself!')
       GOTO 9350
C
 9210  WRITE (1,9211)
 9211   FORMAT (' SERIAL POLL ADDRESS ERROR - no more than one secondary
     _ address')
 9220  WRITE (1,9221)
 9221   FORMAT ('                                  may follow a primary addre
     _ss')
       GOTO 9350
C
 9240  WRITE (1,9241)
 9241   FORMAT (' SERVICE REQUEST - a 488 device is requesting service')
       GOTO 9350
C
 9260  WRITE (1,9261)
 9261   FORMAT (' TIMEOUT ERROR - the specified amount of time has elaps
     _ed without')
 9270  WRITE (1,9271)
```

```
 9271   FORMAT ('                completing a 488 handshake cycle')
        GOTO 9350
C
 9290  WRITE (1,9291)
 9291   FORMAT (' ATN TRUE - an external controller is trying to issue a
     _command')
        GOTO 9350
C
 9310  WRITE (1,9311)
 9311   FORMAT (' IFC TRUE - reset 488 interface')
        GOTO 9350
C
 9330  WRITE (1,9331)
 9331   FORMAT (' S-100 RESET')
        GOTO 9350
C
 9350  CONTINUE
 9360  RETURN
C
 9370  WRITE (1,9371)
 9371   FORMAT (' SYSTEM ERROR - an illegal error code has been encounte
     _red')
 9380  RETURN
       END
C
C  ---------------------------------------- STRXFR --------
C   String Transfer Routine
C
       SUBROUTINE STRXFR (ARRAY,SIZE)
C
       INTEGER I,J,SIZE,STRLEN,ADDR
       BYTE ARRAY
       DIMENSION ARRAY(SIZE)
C
       STRLEN=IUNSGN(ARRAY(1))
       ADDR=256*IUNSGN(ARRAY(3))+IUNSGN(ARRAY(2))
C
       J=STRLEN+3
       IF (SIZE.GE.J) GOTO 9600
       WRITE (1,9505)
 9505  FORMAT (' THE STRING RECEIVED IS BIGGER THAN THE ARRAY GIVEN!!')
       WRITE (1,9506) SIZE
 9506  FORMAT (' Only the first ',I3,' characters were transferred.')
C
       DO 9520 I=4,SIZE
          ARRAY(I)=PEEK(ADDR)
          ADDR=ADDR+1
 9520  CONTINUE
       RETURN
C
C
 9600  DO 9610 I=4,J
          ARRAY(I)=PEEK(ADDR)
          ADDR=ADDR+1
 9610  CONTINUE
C
       J=J+1
       DO 9620 I=J,SIZE
```

```
            ARRAY(I)=0
 9620   CONTINUE
        RETURN
        END
C
C  ---------------------------------------- IUNSGN --------
C   Signed Byte to Unsigned Integer Converter
C
        INTEGER FUNCTION IUNSGN(M)
C
        BYTE M
        INTEGER IUNSGN
C
        IUNSGN=M
        IF (IUNSGN.LT.0) IUNSGN=IUNSGN+256
        RETURN
        END
C
C  ---------------------------------------- STRSET --------
C   String Descriptor Setup Routine
C
        SUBROUTINE STRSET(ARRAY,SIZE,STRLEN)
C
        INTEGER I,J,SIZE,STRLEN
        BYTE ARRAY
        DIMENSION ARRAY(SIZE)
C
C   Set up string descriptor: length, address low, high
C   in the first three bytes of the array
C
        ARRAY(1)=STRLEN
C
C   Get the array address and add an offset to point around
C   the string descriptor
C
        I=IVARPT(ARRAY)+3
C
        J=I/256
        ARRAY(2)=I-256*J
        ARRAY(3)=J
        RETURN
        END
```

The following assembler program is used by FCLOCK.FOR to get the address of a variable. Microsoft Fortran is similar to Basic in that it passes the addresses of parameters, and like Basic, it passes the address of the first parameter in register pair HL. Values returned by functions are put into register pair HL before the function returns to the calling program. Since the address of the parameter of IVARPT is placed in HL by the calling program, and IVARPT immediately returns to the calling program, the value that is returned is the address of the parameter of IVARPT.

```
;
;  PROGRAM IVARPT
;
;  THIS PROGRAM IS DESIGNED TO BE USED AS A
;  FORTRAN FUNCTION CALL SIMILAR TO MICROSOFT
;  VARPTR FUNCTION IN THEIR BASIC.
;
```

```
;  WHEN THE ROUTINE IS CALLED, FORTRAN WILL
;  PASS THE POINTER TO THE ARGUMENT VARIABLE
;  IN THE HL REGISTER PAIR.  SINCE FORTRAN
;  EXPECTS INTEGER FUNCTIONS TO PLACE THE
;  RETURN ARGUMENTS IN THE HL PAIR, ALL THAT
;  NEED BE DONE IS A RET.
;
        PUBLIC IVARPT
;
IVARPT: RET
;
        END
```

**** QCCLOCK.C ****

This program illustrates how MSOFT can be used with a program written in C. The particular compiler used in this example is Q/C written by Quality Computer Systems. It has the advantage of being inexpensive (under $100), readily available, and it can be used with a linker (such as Microsoft's L80).

Like all the other clock programs, this one performs the same function but is written in a different language. It initializes the 488 bus by sending an Interface Clear (IFC), puts the clock into the remote mode by making the REN line true and then addressing the clock as a listener. It then addresses the clock as a talker and listens to the data (status, date and time) that the clock sends over the bus. It displays the date and time each time the minutes change. It also displays the data each time the status character indicates a clock error.

There are a few points you should keep in mind if you use this program as a guide for writing a program for some other C compiler. One is that even though all pass the parameters on the stack, some reverse the order. Q/C places the leftmost parameter on the stack first and the rightmost last, so that the first parameter popped off of the stack is the rightmost one. Some other compilers put the rightmost parameter on the stack first, so that the first parameter popped off of the stack is the leftmost one.

Another potential source of difficulty is that Q/C passes an argument back to the calling procedure in register pair HL. I have no idea of whether other compilers do also. I made use of Q/C's convention in the procedure scntic.

The following dialog shows how to compile this program and link it with MSOFT.REL. The result is an executable file named QCCLOCK.COM.

```
B>CC QCCLOCK.C -M<CR>
QC Compiler V1.01 Copyright (c) 1981 Quality Computer Systems
0 error(s) found
B>M80 =QCCLOCK<CR>

No  Fatal error(s)

B>L80 QCCLOCK,CRUNLIB,MSOFT,QCCLOCK/N/E<CR>

Link-80  3.42  19-Feb-81  Copyright (c) 1981 Microsoft

Data    0103    2A08    <10501>

29959 Bytes Free
[0103    2A08       421

B>
```

```
/*****************************************************************/
/*                                                               */
/*              QCCLOCK.C       4-14-82                          */
/*          for Q/C Compiler  Version 1.01                       */
/*                                                               */
/*    This is a C program which addresses an HP 59309A clock as  */
/* a talker and then reads the time and date.  It continually    */
/* rereads the time and displays the time and date on the        */
/* console each minute.                                          */
/*                                                               */
/* The program assumes that the bus output format of the 59309A  */
/* is set to SPACE, CAL, and COLON.  It also assumes that the     */
/* TALK address of the clock is "E".                             */
/*                                                               */
/*      The Q/C Compiler is distributed by                       */
/*              The Code Works                                   */
/*              Box 550                                          */
/*              Goleta, CA  93116                                */
/*              (805) 683-1585                                   */
/*****************************************************************/

extern  CNTL, CNTLC, TALK, TALKC, LSTN, LSTNC, SPOLL;
extern  PPOLL, DREN, REN, STATUS, IFC, BRSET;
extern  IOSET, PROTCL, ECHO, IOPORT;


#include "qstdio.h"
#define LINLEN  132     /* maximum input line size */
#define LF      10      /* ASCII code for line feed */

/* ------------------------------------ MAIN -----------------*/
main()
  {
/*  Make all the communication variables static because Q/C
    accesses static variables more rapidly and with less object
    code than it accesses automatic variables                  */

  static ercode,        /* error code */
         time,          /* timeout */
         eot,           /* EOT switch */
         eos,           /* EOS byte */
         length,        /* listen string length */
         poll,          /* poll response */
         echoin,        /* input echo switch */
         echout,        /* output echo switch */
         bus;           /* 488 status */

  int   status; /* HP clock status */
  int   umin;   /* units digit of minutes */
  int   oldmin; /* previous units digit of minutes */

  char  line[LINLEN];   /* listen input line */

/*  Pass variable names that CCLOCK will be using to MSOFT     */
  sioset(&ercode, &time, &poll, &bus);
  sprotcl(&eot, &eos, &length);
  secho(&echoin, &echout);
```

```
/*  TIME contains the amount of time to allow for handshake     */
/*  If TIME=255, then the handshake is not timed                */
  time=255;
  echoin=0;      /* turn off input echo */
  echout=0;      /* turn off output echo */

/*  Set up MSOFT so it will stop on EOS (End-of-String) byte,   */
/*  set the EOS byte to be a line feed.                         */
  eot=1;
  eos=LF;

for (;;) {
    ifc();       /* initialize the 488 bus */
    ren();       /* make REN true */

  /*  Become the 488 controller and issue UNLISTEN, UNTALK and  */
  /*  then address the clock as a listener                      */
    scntlc("?_%");

  /*  Report any 488 errors that may have occurred      */
    errmsg(ercode);

  /*  Intialize OLDMIN to some value which cannot match the     */
  /*  first reading from the clock.  This will insure that the  */
  /*  time and date will be displayed the first time through.   */
    oldmin = -1;

    for (;;) {
        /*  Become the 488 controller and issue UNLISTEN, UNTALK and
*/
        /*  then address the clock as a talker
*/
          scntlc("?_E");

        /*  Report any 488 errors that may have occurred        */
          errmsg(ercode);
          if (ercode != 0) break;  /* break out of loop if bus error
*/

        /*  Become a listener and read the clock's status and time
*/
          slstnc(line);

        /*  Report any 488 errors that may have occurred        */
          errmsg(ercode);
          if (ercode != 0) break;  /* break out of loop if bus error
*/

        /*  Check for clock error and report any other errors   */
        /*  (Remember that the first character of the line is   */
        /*   a "?" if the clock is in error.)                   */
          if (line[0] == '?') {clkerr(); puts(line); break;}

        /*  Show the time if the minutes have changed   */
          if (line[12] != oldmin) puts(line);
          oldmin = line[12];    /* update oldmin */
        }
 /*  Check for clock error.  Exit to operating system  */
```

```
    /*  on a clock error because the clock must be reset. */
      if (line[0] == '?') {puts("Reset clock"); break;}
      }
  /* end main */ }

  /* ------------------------------------ CLKERR ----------------*/
  /*   Clock error report routine                                 */
  clkerr()
        {
        fputs("CLOCK ERROR  ",stdout);
        }

  /* ------------------------------------ ERRMSG ----------------*/
  /*   488 interface error reporting routine                      */
  errmsg(code)
  int code;
  {
  if (code != 0) printf("Error Code = %d\n",code,2);

  /* Interpret Error codes and print error messages  */

  if (code & 1) {
    puts("SETUP ERROR - either IOSET or PROTCL was not called before");
    puts("              using one of the MSOFT communication functions");
              }
  if (code & 2)
    puts("NO LISTENERS - I cannot talk to myself!");

  if (code & 4) {
    puts("SERIAL POLL ADDRESS ERROR - no more than one secondary address");
    puts("                            may follow a primary address");
              }
  if (code & 8)
    puts("SERVICE REQUEST - a 488 device is requesting service");

  if (code & 16) {
    puts("TIMEOUT ERROR - the specified amount of time has elapsed
  without");
    puts("                completing a 488 handshake cycle");
              }
  if (code & 32)
    puts("ATN TRUE - an external controller is trying to issue a command");

  if (code & 64)
    puts("IFC TRUE - reset 488 interface");

  if (code & 128)
    puts("S-100 RESET");

  /* end errmsg */ }

  /* ------------------------------------ SCNTLC ----------------*/
  /*   Send a string as a controller                              */
  scntlc(string)
  char string[];
  {
  makedope(string);       /* HL = dope vector address */
  cntlc();                /* clear the error code and send the string */
```

```
}

/* ------------------------------------ SLSTNC ---------------*/
/*   Get a string as a listener                                */
slstnc(string)
char string[];
{
#asm
        POP     B       ;remove return address
        POP     H       ;get addr of beginning of string area
        PUSH    H
        PUSH    B
        PUSH    H       ;addr of beginning of string area
        LXI     H,DOPVTR ;HL points to the dope vector for MSOFT
        CALL    LSTNC   ;clear the error code and get the string
;
;  Copy MSOFT string into the C string.  Remove all carriage returns.
;
        LXI     H,DOPVTR ;HL points to the dope vector
        POP     D        ;DE points to C string area
        MOV     C,M     ;C = MSOFT string length
        INX     H
        MOV     A,M
        INX     H
        MOV     H,M
        MOV     L,A     ;HL = MSOFT string address
CPYSTR: DCR     C
        JM      ENDSTR  ;..no more characters to copy
        MOV     A,M
        INX     H
        CPI     13      ;carriage return?
        JZ      CPYSTR  ;..yes, so do not copy into C string
        STAX    D       ;..no, so copy into C string
        INX     D
        JMP     CPYSTR

ENDSTR: SUB     A       ;terminate C string with a null
        STAX    D
        RET
#endasm
/* end slstnc */}

/* ------------------------------------ MAKEDOPE -------------*/
/*   Make a Microsoft type of string dope vector               */
makedope(string)
char string;
{
#asm
;  C stores a string as a sequence of characters terminated by a NULL.
;  We have to generate a string in the form that MSOFT expects.  This
;  involves two steps.  The first is to generate a "dope vector" which
;  consists of three bytes.  The first is the length of the string, and
;  the second and third are a word which contains the address of the
;  string itself.  Since C passed the address of the string to this
;  routine, all that we really need to do is generate a dope vector.
;
        POP     B       ;get return address out of the way
        POP     H       ;get the address of the string
```

```
            PUSH    H
            PUSH    B       ;put return address on stack
            SHLD    DOPVTR+1 ;put string address in the dope vector
            MVI     C,-1    ;preset the string length counter
SLEN:       INR     C
            SUB     A       ;zero reg A
            ORA     M       ;see if the byte is NULL (zero)
            INX     H       ;point to the next byte
            JNZ     SLEN    ;..try next byte
            MOV     A,C     ;get the string length
            STA     DOPVTR  ;put it in the dope vector
            LXI     H,DOPVTR ;HL = dope vector
            RET             ;return dope vector address

DOPVTR:     DB      0       ;count byte
            DW      0       ;address of string
#endasm
/* end makedope */ }

/* ------------------------------------ SIOSET ----------------*/
/*   Set up communication variables                            */
sioset(ercode,time,poll,bus)
int *ercode, *time, *poll, *bus;
{
#asm
;  Call IOSET with
;       HL = address of ercode
;       DE = address of time
;       BC = address of a table (B reg buffer)
;  The table must contain the following entries:
;  1.   *poll
;  2.   *bus
;  Set DE to point to last parameter passed
            LXI     H,2     ;skip over the return address
            DAD     SP      ;HL = stack pointer upon entry
            MOV     E,M
            INX     H
            MOV     D,M
            INX     H
            XCHG            ;HL = address of bus variable
            SHLD    BBFR2   ;save it in second entry of B reg buffer
            XCHG
            MOV     E,M
            INX     H
            MOV     D,M
            INX     H
            XCHG            ;HL = address of poll variable
            SHLD    BBFR1   ;save it in first entry of B reg buffer
            XCHG
            MOV     E,M
            INX     H
            MOV     D,M
            INX     H       ;DE = address of time
            MOV     A,M
            INX     H
            MOV     H,M
            MOV     L,A     ;HL = address of ercode
            LXI     B,BBFR1 ;point BC to B register buffer
```

```
        JMP     IOSET

BBFR1:  DW      0       ;B register buffer
BBFR2:  DW      0
#endasm
/* end sioset */ }

/* ------------------------------------ SPROTCL ---------------*/
/*  Set up more communication variables                        */
sprotcl(length,eos,eot)
int *length, *eos, *eot;
{
#asm
;  Call PROTCL with
;       HL = address of length
;       DE = address of eos
;       BC = address of eot
;  Set HL to point to last parameter passed
        LXI     H,2     ;skip over the return address
        DAD     SP      ;HL = stack pointer upon entry
        MOV     C,M     ;get address of eot switch
        INX     H
        MOV     B,M     ;BC = address of eot switch
        INX     H
        MOV     E,M
        INX     H
        MOV     D,M     ;DE = address of eos byte
        INX     H
        MOV     A,M
        INX     H
        MOV     H,M
        MOV     L,A     ;HL = address of listen string length
        JMP     PROTCL
#endasm
/* end sprotcl */ }

/* ------------------------------------ SECHO ----------------*/
/*      Set up echo switches                                   */
secho(echoin,echout)
int *echoin, *echout;
{
#asm
;  Call ECHO with
;       HL = address of echoin
;       DE = address of echout
        POP     B       ;hang on to return address
        POP     D       ;DE = address of echout switch
        POP     H       ;HL = address of echin switch
        PUSH    H       ;restore stack
        PUSH    D
        PUSH    B
        JMP     ECHO
#endasm
/* end secho */ }
```

UNOFFICIAL PHRASEBOOK
IEEE 488 to ENGLISH

IEEE used the following conventions when they assigned the names used in the standard:

Lower Case names are associated with local messages (messages between a device and its interface; they MIGHT NOT appear on the 488 bus).

Upper Case names are divided into three groups:

One or two letters name interface functions,

Three letter mnemonics are remote messages (communications over the 488 bus from one interface to another) and

Four letter names ending in "S" identify the state of an interface function.

The numbers following an entry are the pages of the IEEE Standard (Apr 4, 1975) which give further information.

ACDS ACcept Data State
21,22

ACG Addressed Command Group - multiline messages (ØØ-ØF Hex) which affect only addressed devices. The messages GTL (Go To Local), SDC (Selective Device Clear), PPC (Parallel Poll Configure) and GET (Group Execute Trigger) operate only on devices in the LADS (Listener Addressed) state. TCT (Take Control) operates on the device in the TADS (Talk Addressed) state.
48,77

ACRS ACceptor Ready State
21,22

Addressed Commands - Commands belonging to the Addressed Command Group (See ACG)
43

AH Acceptor Handshake - the device function which allows proper reception of data and commands appearing on the eight data lines of the 488 bus (i.e., multiline messages). The DAV (Data Available) line is sensed to determine when the multiline message is valid, and the AH function indicates its readiness for data by asserting a passive false on the NRFD (Not Ready For Data) line, and that it has received the message by asserting a passive false on the NDAC (Not Data Accepted) line. Note that it is illegal for the AH to assert both NDAC and NRFD passive false simultaneously.
20

Active False - an active false message asserted on the 488 bus is one in which it is guaranteed that a false value is received. It overrides a passive true. The standard is constructed so that it is not possible for an active true and an active false message to be asserted on the bus at the same time.
16

Active True - a message which when asserted on the 488 bus is guaranteed to be received as true. It overrides a passive false. The standard is constructed so that it is not possible for an active true and an active false message to be asserted on the bus at the same time.
16

AIDS ACceptor Idle State
20, 21

ANRS Acceptor Not Ready State
20,21

APRS Affirmative Poll Response State
32

ATN ATtentioN - a uniline remote message indicating that a Controller is sending commands (as contrasted to a Talker sending data) over the eight data (DIO) lines.
19,21,24,29,35,41,48,75-76

AWNS Acceptor Wait for New cycle State
21,22

C Controller interface function - the interface function which allows a device to send device addresses, universal commands and addressed commands over the 488 bus. It also allows the device to conduct a Parallel Poll to determine which device needs service.
41

CACS Controller ACtive State
41,42

CADS Controller ADdressed State
41,42

CAWS Controller Active Wait State
41,43

CIDS Controller IDle State
41

CPPS Controller Parallel Poll State
41,43

CPWS Controller Parallel poll Wait State
41,43

CSBS Controller StandBy State
41,43

CSNS Controller Service Not requested State
41,44

CSRS Controller Service Requested State
41,44

CSWS Controller Synchronous Wait State
41,43

CTRS Controller TRansfer State
41,44

DAB DAta Byte - a multiline sent by the Source Handshake (SH) over the eight data (DIO) lines
25,48,75-76

DAC Data ACcepted - the complement appears on the NDAC line. See AH, SH for further information.
19,22,48,75-76

Data Byte Transfer Control lines - the three lines (DAV, NRFD and NDAC) that are used by the Source and Acceptor functions to perform the handshake cycle.
12,18-22,67

DAV DAta Valid - a uniline message sent by the Source Handshake (SH) function over the DAV line. See SH.
48,75-76

DC Device Clear interface function - the interface function which allows a device to be cleared (initialized) either individually or as part of a group. The group may be either part or all of the addressed devices in one system.
37-38

DCAS Device Clear Active State
38

DCIS Device Clear Idle State
37,38

DCL Device CLear - a multiline message (14 Hex) sent by the Controller over the eight data lines indicating that all devices are to go into the Clear state. The details are device dependent, but usually the device is left in the same state as when its power is first turned on.
38,43,48,75-77

Dense Subset - A subset of the Primary Command Group, consisting of only the Listen Address Group (LAG) and Talk Address Group (TAG). ISO codes Space through Underline, inclusive. (Values 20 Hex through 5F Hex).
77

DIOn Data Input/Output line n (n goes from 1 through 8)
54

DT Device Trigger interface function - the interface function which allows a device to start its basic operation started either individually or as part of a group. This function may be used to start several devices simultaneously.
38-39

DTAS Device Trigger Active State
39

DTIS Device Trigger Idle State
39

END END - a uniline message sent by a Talker (EOI line active true) at the same time a data byte is sent on the data (DIO) lines. The message indicates that this is the last data byte to be sent. (See EOS for an alternate way of terminating a string sent by a Talker).
23,48,75-76

EOI End Or Identify - a uniline message which serves two purposes: if asserted true by a Talker it indicates that the last byte of a string is being sent. If asserted true by a Controller it initiates a Parallel Poll.

EOS End Of String - a multiline message sent by a Talker to indicate that the last byte of a string has been sent. Its value (ISO code) is determined by what the Listener(s) recognize.
48

General Interface Management lines - the five lines used to perform system operations, such as Parallel Poll, Interface Clear, etc. Several of the lines are also used in data transactions: an example is EOI, which may be used to signal the end of a multibyte transaction. The five lines are ATN, EOI, IFC, REN and SRQ.
12

GET Group Execute Trigger - a multiline message (Ø8 Hex) sent by the Controller indicating that all devices addressed as Listeners are to start performing their respective functions. This command is often used to start several pieces of equipment in synchronism.
39,43,48,75-77

GTL Go To Local - a multiline message (Ø1 Hex) sent by the Controller indicating that all devices addressed as Listeners are to go to the Local state: i.e., local controls on the front or back panel (instead of device dependent messages on the 488 bus) control device operation. (See Local Control)
33,43,48,75-77

gts go to standby - a local message sent by a device to its Controller interface function telling it that it is finished sending commands. The response is that the Controller function releases the bus so that other operations (e.g., a Talker sending data to Listeners) may proceed.
41,75

IDY IDentifY - a uniline message sent by the Controller during a Parallel Poll telling the other devices to assert their Parallel Poll responses on the data bus.
35,48,75-76

IFC InterFace Clear - a uniline message sent by the System Controller telling all other devices on the bus to go to the Idle state. This message is used to place all devices in a known state. It should be used sparingly because any bus transaction is terminated by this function.
24,29,41-42,48,75-76

ISO Code - a seven bit code equivalent to the American National Code for Information Interchange, ANSI X3.4-1968 (often called ASCII).
46,50,77

isr individual service request - a local message sent by a device to its Parallel Poll interface function. If the individual status (see "ist") message is equal to the S (Sense) bit received as part of the most recently received PPE (Parallel Poll Enable) command, the PPR (Parallel Poll Response) byte specified by the three bits P1-P3 of the most recent PPE command must be sent true upon receipt of an IDY (Identify) command from the Controller. Alternately, if subset PP2 (Parallel Poll function cannot be configured by the Controller) is used, local messages are substituted for S, P1-P3.
35-37,75

ist individual status - a local message used by the Parallel Poll function to determine the proper response to an IDY (Identify) command from the Controller. See "isr".
35-36

L Listen interface function - the function which allows a device to receive data from the 488 bus.
28

LACS Listener ACtive State
29-30

(LAD) the listen address of a specific device (received as MLA). See "MLA".
43

LADS Listener ADdressed State
28-29

LAG Listen Address Group - a subset of the ISO-7 codes, being characters SPACE through ? (20 Hex through 3F Hex).
48, 77

LE Listen Extended interface function - similar to the Listen function except that a Secondary Address must be used as well as the Primary Address used for the Listen function.
30

LIDS Listener IDle State
28-29

LLO Local LockOut - a multiline command (11 Hex) sent by the Controller which tells all devices with the RL (Remote Local) interface function to obey device dependent messages sent over the 488 bus instead of their local controls (e.g., front panel).
33,43,48,75-77

LOCS LOCal State
33

local control - the device is programmed by its controls instead of by the 488 interface. An example is a digital multimeter; the range, function, sample rate, etc. are set by front panel controls if it is under local control.
33

local message - a message sent between a device function and an interface function. It may cause a remote message to be sent from the interface function over the 488 bus.
15

lon listen only - a local message which causes the Listen function of the device to act as if it had been addressed by the Controller.
29,75

LPAS Listener Primary Addressed State
29,30

lpe local poll enable - a local message which causes the Parallel Poll function of the device to act as if it has received a PPE (Parallel Poll Enable) from the Controller. When lpe is false, the device is to act as if it has received a PPD (Parallel Poll Disable) while in the PACS (Parallel Poll Addressed to Configure state) or a PPU (Parallel Poll Unconfigure) command from the Controller.
35,75

LPIS Listener Primary Idle State
29-30

ltn listen - a local message which when true and the Controller is in the active state causes the L (Listen) or LE (Listen Extended) function to go from the Idle (LIDS) to the Addressed (LADS) state.
29,75

lun local unlisten - a local message which when true and the Controller is in the active state (CACS) causes the L (Listen) or LE (Listen Extended) function to go from the Addressed (LADS) to the Idle (LIDS) state.
29,75

LWLS Local With Lockout State
33-34

MLA My Listen Address - the address which the L (Listen) or LE (Listen Extended) function will respond to. Note that the standard does not allow a 488 bus system to have both an L and an LE interface function which respond to the same primary address. MLA must belong to the LAG (Listen Address Group).
48,75-76

MSA My Secondary Address - the secondary address which the TE (Talk Extended) or LE (Listen Extended) functions will respond to if they are in the Primary Addressed state (TPAS or LPAS, respectively). MSA must belong to the SCG (Secondary Command Group).
24,48,75-76

MTA My Talk Address - the primary address which the T (Talk) or TE (Talk Extended) function will respond to. Note that the standard does not allow a 488 bus system to have both a T and TE interface function simultaneaously with the same primary address. MTA must belong to the TAG (Talk Address Group).
24,29,48,75-76

multiline message - a message that is sent over two or more lines of the 488 bus. An example is Device Clear (DCL) (14 Hex sent out on the data (DIO1-DIO8) lines by the Controller).
45

nba new byte available - a local message sent by a device to its Source Handshake (SH) function to inform it that another byte is available for it to place on the bus data (DIO1-DIO8) lines.
19,75

NDAC Not Data ACcepted - one line of the 488 bus which carries the complement of the Data ACcepted (DAC) message. It is one of the three Data Byte Transfer Control lines. (See DAC).

NPRS Negative Poll Response State
32

NRFD Not Ready For Data - one line of the 488 bus. It carries the complement of the Ready For Data (RFD) message, and is one of the three Data Byte Transfer Control lines. (See RFD).

NUL null byte: all eight bits are false.
23,42,48

OSA Other Secondary Address - a secondary address which is not the same as the secondary address of the TE (Talk Extended) function while it is in the TPAS (Talk Primary Addressed state), or of the LE (Listen Extended) function while it is in the LPAS (Listen Primary Addressed state). OSA must belong to the SCG (Secondary Command Group).
48,75-76

OTA Other Talk Address - an address other than a device's own talk address. Some devices which are capable of talking unaddress themselves if they sense that the Controller is addressing another Talker. This feature can be convenient because an UNTalk (UNT) command is not needed. OTA must belong to the TAG (Talk Address Group).
24,48,75-76

PACS Parallel poll Addressed to Configure State
35-36

Passive False - a message which when asserted on the 488 bus is NOT guaranteed to be received as false. It is overridden by an active true message.
16

Passive True - a message which when asserted on the 488 bus is NOT guaranteed to be received as true. It is overridden by an active false message.
16

PCG Primary Command Group - a subset of the ISO-7 code. It consists of all characters NUL through UNDERLINE (ØØ Hex through 5F Hex). It includes all of the ACG (Addressed Command Group), UCG (Universal Command Group), LAG (Listen Address Group) and TAG (Talk Address Group).
35,49,75-77

pon power on - a local message sent by the device to its own interface to inform it that power has just been applied. The interface should reset all functions (e.g., Listen, AH, Talk, etc.) to their idle states.
75

PP Parallel Poll interface function - the function which allows a device to respond to a Parallel Poll from the Controller.
35

PPAS Parallel Poll Active State
35-36

PPC Parallel Poll Configure - a multiline message (Ø5 Hex) sent by the Controller which causes the device presently addressed as a Listener (e.g., in the LADS state) to go into the PACS (Parallel Poll Addressed to Configure) state. While in the PACS, the PP (Parallel Poll) function is to obey the PPE (Parallel Poll Enable) and PPD (Parallel Poll Disable) messages sent by the Controller.
35,43,75-77

PPD Parallel Poll Disable - a multiline message (7Ø Hex) sent by the Controller which will place all devices in the PACS (Parallel Poll Addressed to Configure) state into the PPIS (Parallel Poll Idle) state.
35,43,49,75-76

PPE Parallel Poll Enable - a multiline message (6Ø-6F Hex) sent by the Controller which will change all devices in the PPIS (Parallel Poll Idle) state to the PPSS (Parallel Poll Standby) state. It also specifies the PPRn (Parallel Poll Response byte) to be used and the S (Sense) of the PPR. The form of the message is (from most significant bit to least)

X 1 1 Ø S P3 P2 P1

where X means don't care (may be either high or low), and the binary value formed by P3-P1 indicates which PPRn is to be used. Note that n of PPRn indicates which data line is to be made active true (i.e., DIO3 will be made active true when PPR3 is placed on the bus).
35,43,49,75-76

PPIS Parallel Poll Idle State
35-36

PPRn Parallel Poll Response n (See PPE)
35,49,75-76

PPSS Parallel Poll Standby State
35-36

PPU Parallel Poll Unconfigure - a multiline message (15 Hex) sent by the Controller which takes all devices in the PPSS (Parallel Poll Standby) state and puts them into the PPIS (Parallel Poll Idle) state.
35,43,49,75-77

PUCS Parallel poll Unaddressed to Configure State
35-36

rdy ready for next message - a local message sent by a device to its AH (Acceptor Handshake) interface function to indicate it is ready for another message byte from the 488 bus (i.e, another multiline remote message).
21,75

remote control - a device is programmed by its 488 interface instead of by local controls. An example is a DMM whose function, range selection, etc are selected by messages sent to it over the 488 bus. See local control for contrast.
33

REMS REMote State
33-34

REN Remote ENable - one of the five General Interface Management lines. Also, a uniline message sent by the Controller to put devices addressed as Listeners into the REMS (Remote) state. When the Controller makes the REN message false, all devices are to go to the LOCS (Local) state.
33,42,49,75-76

RFD Ready For Data - the complement appears on the NRFD line. This uniline message is used by the AH (Acceptor Handshake) function to indicate that it is ready to accept the next byte (multiline message). See AH for further information.
19,22,49,75-76

RL Remote Local interface function - if present it allows a device to be switched from local to remote control and vice versa.
33

rpp request parallel poll - a local message sent to the Controller interface function when the device wants a Parallel Poll performed.
41,75

RQS ReQuest Service - the byte sent by the current Talker in response to a Serial Poll. Data bit 7 (DIO7) is true.
23,49,75-76

rsc request system control - a local message sent to the Controller interface function by the device when it wants to go to the SACS (System Control Active) state.
41,75

rsv request service - a local message sent by a device to its Service Request interface function to cause it to go to the SRQS (Service Request) state. As a consequence, the uniline message SRQ is sent active true until either rsv is sent false, or the Controller performs a Serial Poll of this device.
32,75

rtl return to local - a local message sent by a device to its Remote/Local interface function. The LOCS (Local) state is entered if neither LLO (Local Lockout) nor ACDS (Accept Data State) are true.
33,75

RWLS Remote With Lockout State
33,34

SACS System Control Active State
41,44

(SAD) Secondary ADdress - the seconday address of a specific device, and is received as either My Seconday Address (MSA) or Other Secondary Address (OSA). Its value must lie in the range 60-7E Hex. (See SCG).
43

(SBA) Status Byte, service request Acknowledged. A message sent over the 488 bus by the current Talker in response to a Serial Poll. This message indicates that this device was requesting service. Data bit 7 (DIO7) is true. (See RQS)
62

(SBN) Status Byte, service Not requested. Same as SBA but indicates that this device does not need service. Data bit 7 (DIO7) is false.
62

SCG Secondary Command Group. A subset of the ISO-7 code consisting of characters ACCENT GRAVE through TILDE (60 Hex through 7E Hex). Secondary Talk and Listen addresses must be selected from this group. (Note that DEL is not allowed as a secondary address).
49, 77

SDC Selected Device Clear - a multiline message (Ø4 Hex) sent by the Controller indicating that all devices addressed as Listeners are to go into the DCAS (Device Clear Active) state. The details are device dependent, but usually the device is left in the same state as when its power is first turned on.
38,43,49,75-77

SDYS Source DelaY State
18-19

Secondary Commands - the commands PPE, PPD and (SAD).
43

SGNS Source GeNerate State
18-19

SH Source Handshake interface function. The function used by a Talker or Controller to insure proper communication of multiline messages. The NRFD and NDAC lines are sensed to determine whether the AH (Acceptor Handshake) function of some device is active (if both NRFD and NDAC are false simultaneously, there is no AH function on the bus, which is an error). The multiline message is placed on the eight data lines (DIO1-DIO8) and a 2 microsecond timeout is started. When NRFD is sensed false and the timeout has been completed (to insure the data lines have settled) DAV is asserted true (to show that the data is available and settled). Upon sensing NDAC false the SH asserts DAV false (to indicate that the data may no longer be valid) then removes the data. The whole cycle is repeated for subsequent bytes of data. (See AH for the other half of the handshake cycle).
18

SIAS System control Interface clear Active State
41,44

sic send interface clear - a local message which causes the devices' Controller interface function to enter the SIAS (System Control Interface Clear Active) state if it is the System Controller (i.e., it is in the SACS (System Control Active) state). As a consequence, the IFC (Inteface Clear) signal is sent active true. (IFC is a uniline message sent on the IFC line).
41,75

SIDS Source IDle State
18-19

SIIS System control Interface clear Idle State
41,44

SINS System control Interface clear Not active State
41,44

SIWS Source Idle Wait State
19-20

SNAS System control Not Active State
41,44

SPAS Serial Poll Active State
24,26

SPD Serial Poll Disable - a multiline message (19 Hex) sent by the Controller. It informs all devices capable of being Talkers that they are to speak data when they are addressed to talk. (See SPE for contrast).
43,49,75-77

SPE Serial Poll Enable - a mulitline message (18 Hex) sent by the Controller. It informs all devices capable of being Talkers that they are to speak their Serial Poll Status Byte (instead of data) when they are addressed to talk. See SBA, SBN, STB for further information about the status byte.
43,49,75-77

SPIS Serial Poll Idle State
24,26

SPMS Serial Poll Mode State
24,26

SR Service Request interface function. This function allows a device to asynchronously request service from the Controller-In-Charge.
31

SRAS System control Remote enable Active State
41,45

sre send remote enable - a local message sent by a device to its Control interface function. It causes the function to enter the SRAS (System Control Remote Enable Active) state only if it was already in the SACS (System Control Active) state. The uniline message REN is sent active true as long as the Controller remains in the SRAS state.
41,75

SRIS System control Remote enable Idle State
41,44

SRNS System control Remote enable Not active State
41,45

SRQ Service ReQuest - a uniline message sent on the SRQ line by the SR (Service Request) interface function. It is the duty of the Controller to provide the service needed.
49,75-76

SRQS Service ReQuest State
32

STB STatus Byte. Data bits 1 through 6 and bit 8 (DIO1-DIO6, DIO8) sent in response to a Serial Poll. STB is combined with RQS to form the complete byte. (See SBA, SBN).
25,49,75-76

STRS Source TRansfer State
18-19

SWNS Source Wait for New cycle State
18-19

T Talk interface function. This function allows a device to send information to other devices on the 488 bus. Only one byte (selected from the Talker Address Group) need be sent to address the Talker.
23

TACS Talker ACtive State
24,26

(TAD) the Talk ADdress of a specific device. It is received as either My Talk Address (MTA) or Other Talk Address (OTA). It must be a member of the TAG (Talk Address Group).
43

TADS Talker ADdressed State
23-24

TAG Talker Address Group. A subset of the ISO-7 code consisting of all characters from @ through UNDERLINE (40 Hex through 5F Hex). The address of a Talker (or the primary address of an Extended Talker) must be selected from this group. Note that UNDERLINE cannot be used as an address, for it is reserved as the Universal Untalk command.
49, 77

tca take control asynchronously - a local message sent by a device to its Controller interface function. It causes the function to go from the CSBS (Controller Standby) state to the CSWS (Controller Synchronous Wait) state, where it waits for at least 500 nsec (to allow the other devices on the 488 bus to respond to the active true assertion of the uniline message ATN), then proceed to the CAWS (Controller Active Wait) state. ATN is active true in both CSWS and CAWS.
41,75

tcs take control synchronously - a local message sent by a device to its Controller interface function. It operates the same as tca EXCEPT that the function goes from CSBS to CSWS only when the AH (Acceptor Handshake) function is in the ANRS (Acceptor Not Ready) state. The effect is to insure that a message sent by a Talker is not garbled or misinterpreted as a message sent by the Controller; ATN will not become active true until the Source Handshake is complete (i.e., DAV is false, showing that the message is no longer valid).
21,41,75

TCT Take ConTrol - a multiline message (09 Hex) sent by the Controller to inform the device currently addressed as a Talker that it is to become the Controller-in-Charge.
41,43,49,75-77

TE Talker Extended interface function. Similar to the Talker (T) function except that this one is addressed by two bytes. The first must be selected from the Talker Address Group (TAG) and the second from the Secondary Command Group (SCG).
23

TIDS Talker IDle State
23-24

ton talk only - a local message sent by a device to its Talk interface function. If IFC (Interface Clear) is false, the Talker function enters the TADS (Talker Addressed) state. Remember that only one Talker may be addressed at a time, so as long as ton is true no other device may have ton true or be addressed as a Talker by the Controller.
24,75

TPAS Talker Primary Addressed State
24,26

TPIS Talker Primary Idle State
24,26

UCG Universal Command Group - A subset of the ISO-7 code consisting of all characters from DLE through US (10 Hex through 1F Hex). These commands operate upon all devices which are capable of responding to a Controller; the devices are not individually addressed. For contrast see Addressed Command Group (ACG).
43,49,77

uniline message - a message that uses only one line of the 488 bus. An example is Service ReQuest (SRQ).

Universal Command Group - See UCG

UNL UNListen - a multiline message (3F Hex or the character "?") sent by the Controller which forces the Listen function of all devices into the LIDS (Listen Idle) state.
29,43,49,75-77

UNT UNTalk - a multiline message (5F Hex or the character "_") sent by the Controller which forces the Talk function of all devices into the TIDS (Talk Idle) state.
49,77

Code Assignments for "Command Mode" of Operation.

(SENT AND RECEIVED WITH ATN TRUE)

Bits b_7 b_6 b_5 (2) / b_4 b_3 b_2 b_1 / COLUMN → / ROW ↓	0 0 0	MSG (1)	0 0 1	MSG	0 1 0	MSG	0 1 1	MSG	1 0 0	MSG	1 0 1	MSG	1 1 0	MSG	1 1 1	MSG
	0		1		2		3		4		5		6		7	
0 0 0 0 0	NUL		DLE		SP		0		@		P				p	
0 0 0 1 1	SOH	GTL	DC1	LLO	!		1		A		Q		a		q	
0 0 1 0 2	STX		DC2		"		2		B		R		b		r	
0 0 1 1 3	ETX		DC3		#		3		C		S		c		s	
0 1 0 0 4	EOT	SDC	DC4	DCL	$		4		D		T		d		t	
0 1 0 1 5	ENQ	PPC (3)	NAK	PPU	%		5		E		U		e		u	
0 1 1 0 6	ACK		SYN		&		6		F		V		f		v	
0 1 1 1 7	BEL		ETB		'		7		G		W		g		w	
1 0 0 0 8	BS	GET	CAN	SPE	(		8		H		X		h		x	
1 0 0 1 9	HT	TCT	EM	SPD	)		9		I		Y		i		y	
1 0 1 0 10	LF		SUB		*		:		J		Z		j		z	
1 0 1 1 11	VT		ESC		+		;		K		[		k		{	
1 1 0 0 12	FF		FS		,		<		L		\		l		|	
1 1 0 1 13	CR		GS		-		=		M		]		m		}	
1 1 1 0 14	SO		RS		.		>		N		^		n		~	
1 1 1 1 15	SI		US		/		?	UNL	O		_	UNT	oo		DEL	

(4)

MLA ASSIGNED TO DEVICE

MLA ASSIGNED TO DEVICE

MTA ASSIGNED TO DEVICE

MTA ASSIGNED TO DEVICE

MEANING DEFINED BY PCG CODE

MEANING DEFINED BY PCG CODE

ADDRESSED COMMAND GROUP (ACG)

UNIVERSAL COMMAND GROUP (UCG)

LISTEN ADDRESS GROUP (LAG)

TALK ADDRESS GROUP (TAG)

PRIMARY COMMAND GROUP (PCG)

SECONDARY COMMAND GROUP (SCG)

NOTES:
(1) MSG = INTERFACE MESSAGE
(2) b_1 = DIO1 . . . b_7 = DIO7
(3) REQUIRES SECONDARY COMMAND
(4) DENSE SUBSET (COLUMN 2 THROUGH 5). ALL CHARACTERS USED IN BOTH COMMAND & DATA MODES.

Courtesy of Hewlett-Packard Co.

P&T-488 Auxilliary Programs for CP/M †

The program BUSMON monitors and reports all transactions on the IEEE-488 bus. 488TODSK records data sent over the 488 bus into a disk file. DSKTO488 sends the contents of a disk file over the bus as data. HANDSHAK.ASM contains the source code for routines which perform the Source and Acceptor Handshake functions. An example of how to use HANDSHAK.ASM is given in the program SAMPLHS.ASM.

BUSMON

The program BUSMON monitors and reports all transactions which occur on the IEEE-488 bus. The operator can choose two different forms for the report. The **normal** form displays the transactions without any special handling. The other form is **expanded**, which means that non-printing characters are replaced with strings of printable characters. This form is especially useful for those cases where one is trying to distinguish between tabs and spaces, or determine whether line feed precedes carriage return, etc. The form of the report can be selected by typing a character on the console keyboard while the program is running. Once the form has been selected, its action may be repeated by typing any key on the keyboard.

The operator can set BUSMON to stop on one of three different conditions: on each carriage return, line feed, or each character. The condition is selected by using one of the four **stop code** keys. The stop code can be changed at any time by typing the appropriate stop code key. The stop code keys and the corresponding stop conditions are shown in the following table. Note that typing a stop code key will NOT cause a repeat of the previous stop condition, but will invoke a new stop condition. The program starts in the Carriage Return mode.

Expand/Normal Option

N or n Show characters normally

X or x Expand the non-printing characters. Space (20 Hex), Horizontal Tab (9) and Line Feed (0A Hex) are replaced by the strings <SPACE>, <HT> and <LF> respectively. The non-printing character Carriage Return (0D Hex) causes the message <CR> to be printed followed by a carriage return and a line feed. All other non-printing characters are replaced with the two character string of an up arrow followed by a capital letter. Thus the non-printing character 01 Hex is replaced by the string ↑A, while the character 1A Hex is printed as ↑Z.

Stop Codes

Carriage Return	Display all transactions up to and including the next carriage return.
Line Feed	Display all transactions up to and including the next line feed.
Space	Display the next transaction (allows stepping one byte at a time).
G or g	Go. Display all transactions continuously without stopping on Line Feed, Carriage Return or next byte.

† CP/M is a trademark of Digital Research

Abort

Control C Abort. Go back to the CP/M command mode.

Console/Printer Switch

Ø Direct all output to the console.
1-9 Direct all output to the system printer.

NOTE: to direct output to both the console and printer, select the console and then press Control P.

IEEE-488 Functions

I or i Assert IFC (perform an Interface Clear).

R or r Make REN true (assert Remote Enable).
L or l Make REN false (all instruments will go to Local mode).

Q or q Make SRQ true (request service).
W or w Make SRQ false (cease requesting service).

P or p Perform a Parallel Poll and report the results.

S or s Show the state of the IEEE-488 lines.

T or t Talk - collect a string of characters from the operator then send it over the bus as a Talker.

C or c Control - collect a string and send it over the bus as a Controller.

NOTE: While collecting a string for Talk or Control the following keys have special meaning:

Control X Delete the string and restart collection. This allows errors to be corrected.

RETURN Terminate the collection of the string. The carriage return is <u>not</u> included in the string.

ESCAPE Put the next character into the string. This allows ESCAPE, RETURN and Control X to be put into the string. For instance, to get the string ?A<ESCAPE>12<RETURN><LINE FEED>, you would type ?A<ESCAPE><ESCAPE>12<ESCAPE><RETURN><LINE FEED><RETURN>. In this example, the string <ESCAPE> means that the ESCAPE key is pressed, not that the 8 keys <, E, S, C, A, P, E and > are pressed. Similarly, <RETURN> and <LINE FEED> mean that the RETURN and LINE FEED keys are used.

Each time the Controller becomes active (asserts ATN active true), a carriage return-line feed is sent to the console, followed by the string **COMMAND:**, followed by another carriage return-line feed pair. Similarly, each time the Controller becomes inactive (ATN is false), a carriage return, line feed, the string **DATA:**, carriage return and a line feed is sent to the console. Thus all characters printed after COMMAND: and before **DATA:** are instructions sent by the Controller, (for example, "?" means

UNLISTEN). All characters printed after DATA: and before COMMAND: are data (otherwise known as device-dependant messages). Examples are readings from a DVM which has been commanded to be a Talker, etc.

Messages are also printed on the console to indicate occurances of IFC (Interface Clear), indicate a change of the state of the REN (Remote Enable) line, and of the SRQ (Service Request) line. The message >>> S-100 POC/RESET TRUE <<< is printed whenever the Power On Clear or the RESET line of the S-100 system becomes true.

Whenever the Controller is active, a descriptive string is substituted for special non-printing messages. For example, >> GO TO LOCAL << is printed when 01 Hex is received and ATN is true. The list of messages and the corresponding non-printing characters is as follows:

Character Hex	Message
01	>> GO TO LOCAL <<
04	>> SELECTIVE DEVICE CLEAR <<
05	>> PARALLEL POLL CONFIGURE <<
08	>> GROUP EXECUTE TRIGGER <<
09	>> TAKE CONTROL <<
11	>> LOCAL LOCKOUT <<
14	>> UNIVERSAL DEVICE CLEAR <<
15	>> PARALLEL POLL UNCONFIGURE <<
18	>> SERIAL POLL ENABLE <<
19	>> SERIAL POLL DISABLE <<

The results of this program can be misleading for the following reasons:

1. This program functions as a Listener on the 488 bus. If there were no Listeners on the bus before this routine was run, any Talker would have been unable to say a thing. However, when this routine is run, the Talker has someone to talk to. Thus the operation of the 488 system may be changed by the fact that the Bus Monitor routine is run.

2. This routine is slow compared to the speed that communication on the 488 bus is capable of attaining. Thus 488 throughput may be drastically slowed by using the bus monitor.

3. This routine is incapable of sensing a Parallel Poll issued by another controller, or the response to that Parallel Poll. If it happens that this routine tests the EOI line at the time of a Parallel Poll, it will show the message <END>, even though ATN is true.

488TODSK

The program **488TODSK** is used to record all data transactions directly into a CP/M disk file. To use the program type

488TODSK filename.ext x<CR>

where **filename.ext** is the file name and extension of the file into which the data is to be recorded, and x is the option code. Note that there must be one and only one space

between 488TODSK and the file name, and also one and only one space between the file name and the option code. The characters <CR> mean that the Carriage Return key is pressed, **not** that the four keys <, C, R and > are pressed.

Three different options are available: none, Z and E. The option E means that the file will be closed and control passed back to the console upon receipt of the 488 END message. The option Z means that the file will be closed and control passed back to the console upon receipt of a Control Z in the data stream (the Control Z is also placed in the file). This option can be useful because CP/M text files are terminated by a Control Z. If no option is selected (that is, a Carriage Return follows the file name), the file can be closed only by pressing Control C on the console. Note that Control C can be used at any time to abort the program: all data received up to the time the Control C was pressed is saved in the file. Some garbage will also appear at the end of the file because the whole buffer is saved in the disk file, and the buffer probably was not filled at the time Control C is pressed.

Error messages are printed on the console if the disk directory is full, the data area is full, or any other disk write error occurs. In each case the function is aborted. If the name of the file is the same as one which is already on the disk, the operator is asked if it is OK to replace the old file. If the operator responds by typing any character other than "Y" or "y", the function is aborted and the old file is left untouched. If the operator responds with either "Y" or "y", the old file is erased and the new one takes its place.

DSKTO488

The program DSKTO488 sends the contents of a CP/M disk file over the 488 bus. The program is called by the string

DSKTO488 filename.ext x

where **filename.ext** is the name of the file that is to be sent and **x** is the option code. Only two options are available: none and Z. The Z option causes the Control Z to be sent with the 488 END message when a Control Z is found in the file, then the program returns control to the console. This can be useful for text files that are terminated by a Control Z. If no option code is selected, the entire file is sent followed by a null with the 488 END message, then control is returned to the console. The program may be aborted at any time by typing Control C on the console.

Error messages are printed on the console if there is no Listener on the bus, if the file is not on the disk, or if an invalid option code is selected. In each case the program is aborted and control is returned to the console.

If you have two systems and want to send a file from one to the other via the 488 bus, you would type

488TODSK filename.ext E<CR>

on the system which is to receive the file, and

DSKTO488 filename.ext<CR>

on the one which is sending the file. (It is not necessary to use the same file name or extension.) Note that the system receiving the file must be started first, otherwise the first byte of the file will be lost or the sending system will complain that there are no listeners.

HANDSHAK

The source file **HANDSHAK.ASM** is actually two subroutines: a routine for Source handshake and a routine for Acceptor handshake. These routines can be useful in special applications where it is desired to use the S-100 system as a Talk Only or Listen Only device, or where increased data rate on the 488 bus is needed. These routines are capable of running much faster than the larger Custom System, CPM488 or 488BAS routines because the larger routines check for the existance of another Controller on the bus, check for excessive time in the handshake cycle, and many other things.

Refer to the chapter titled **Hardware Description** in the P&T-488 manual for information about the bit mapping of the ports and the 488 bus lines.

SAMPLHS

This file contains the source code for a routine which uses the Source, Acceptor and Initialization subroutines in HANDSHAK to take data from the IEEE-488 bus and display it on the console.

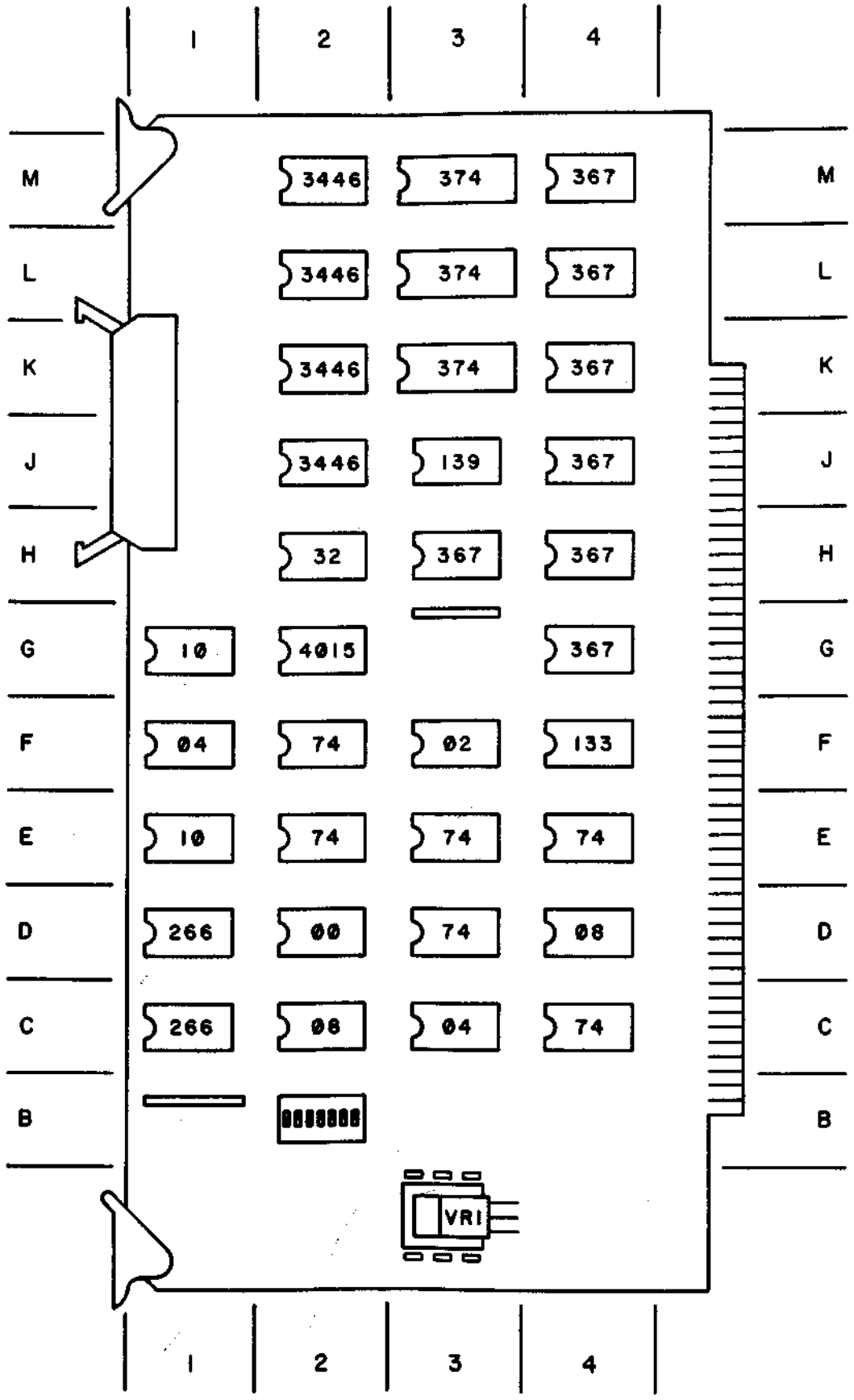
```
;*****************************************************************
;
;       Source and Acceptor Handshake listings
;
;*****************************************************************

ISRPT   EQU     7CH
CMDPT   EQU     ISRPT+1
DATPT   EQU     ISRPT+2
PPORT   EQU     ISRPT+3

MONITR  SET     0       ;CP/M warmstart entry
CPMIO   SET     5       ;CP/M I/O entry point

CR      SET     0DH     ;ASCII  carriage return
LF      SET     0AH     ;ASCII  line feed
ES      SET     '$'     ;CP/M buffered print string terminator

BUFPRN  SET     9       ;CP/M fcn. number for buffered print
;
;       TALK
;
TLKT:   LDA     GIMTC   ;get the image of the byte last sent
                        ;  to the command line port
        ORI     8       ;make sure that ATN is false (high)
        STA     GIMTC   ;  when do source handshake
;
;*****************************************************************
;
;       SOURCE HANDSHAKE
;
;  This routine takes the byte in memory location CHAR and says
;  it on the 488 bus as a Talker.  If either the S-100 RESET
;  or Power On Clear line is or has been true, or if the
;  488 ATN or IFC lines are or have been true, then an error
;  message is printed and the routine jumps to the system
;  monitor.
;
;*****************************************************************
;
SRCHS:  LDA     GIMTC   ;get 488 command line image
        ORI     60H     ;set NRFD, NDAC high (false)
        CALL    COMND
SRC1:   CALL    INTRPT  ;check for POC, ATN or IFC
        JNZ     BYE     ;..abort if POC, ATN or IFC true
        IN      CMDPT   ;see if there are any listeners
        CMA
        ANI     60H     ;check only NRFD, NDAC
        JZ      NOLSN   ;..no listeners error
        ANI     40H     ;wait until NRFD is high (false)
        JNZ     SRC1
        LDA     CHAR    ;get the data byte.
        CMA             ;488 uses negative logic
```

```
        OUT     DATPT
        LDA     GIMTC
        ANI     7FH     ;make DAV true (low)
        CALL    COMND
SRC2:   CALL    INTRPT  ;check for POC, ATN or IFC
        JNZ     BYE     ;..abort if POC, ATN or IFC true
        IN      CMDPT
        ANI     20H     ;look at NDAC line
        JZ      SRC2    ;...data not accepted yet
        LDA     GIMTC
        ORI     81H     ;make DAV & EOI false (high)
        CALL    COMND
        MVI     A,0FFH
        OUT     DATPT   ;make all data lines passive false
        RET
;
;*****************************************************************
;
;       ACCEPTOR HANDSHAKE
;
;  This routine gets one byte from the 488 bus and returns with
;  it in register A.  If either the S-100 RESET or Power On
;  Clear line is or has been true, or if the 488 ATN or IFC
;  lines are or have been true, then an error message is printed
;  and the routine jumps to the system monitor.
;
;*****************************************************************
;
ACEPTR: LDA     GIMTC
        ORI     8       ;make ATN false
        ANI     9FH     ;  and NRFD true, NDAC true
        CALL    COMND
        LDA     GIMTC
        ORI     40H     ;now make NRFD false
        CALL    COMND
ACEPT1: CALL    INTRPT  ;see if received POC, ATN or IFC
        JNZ     BYE     ;..abort
        IN      CMDPT   ;look at DAV
        ANI     80H
        JNZ     ACEPT1  ;..DAV still false
        IN      DATPT   ;get the data
        CMA             ;488 uses negative logic
        MOV     D,A     ;keep the data in register D
        LDA     GIMTC
        ORI     20H     ;NDAC false
        ANI     0BFH    ;NRFD true
        CALL    COMND
ACEPT2: CALL    INTRPT
        JNZ     BYE     ;..abort
        IN      CMDPT   ;wait for DAV false
        ANI     80H
        JZ      ACEPT2  ;...DAV still true
        LDA     GIMTC
        ANI     9FH     ;NRFD true, NDAC true
```

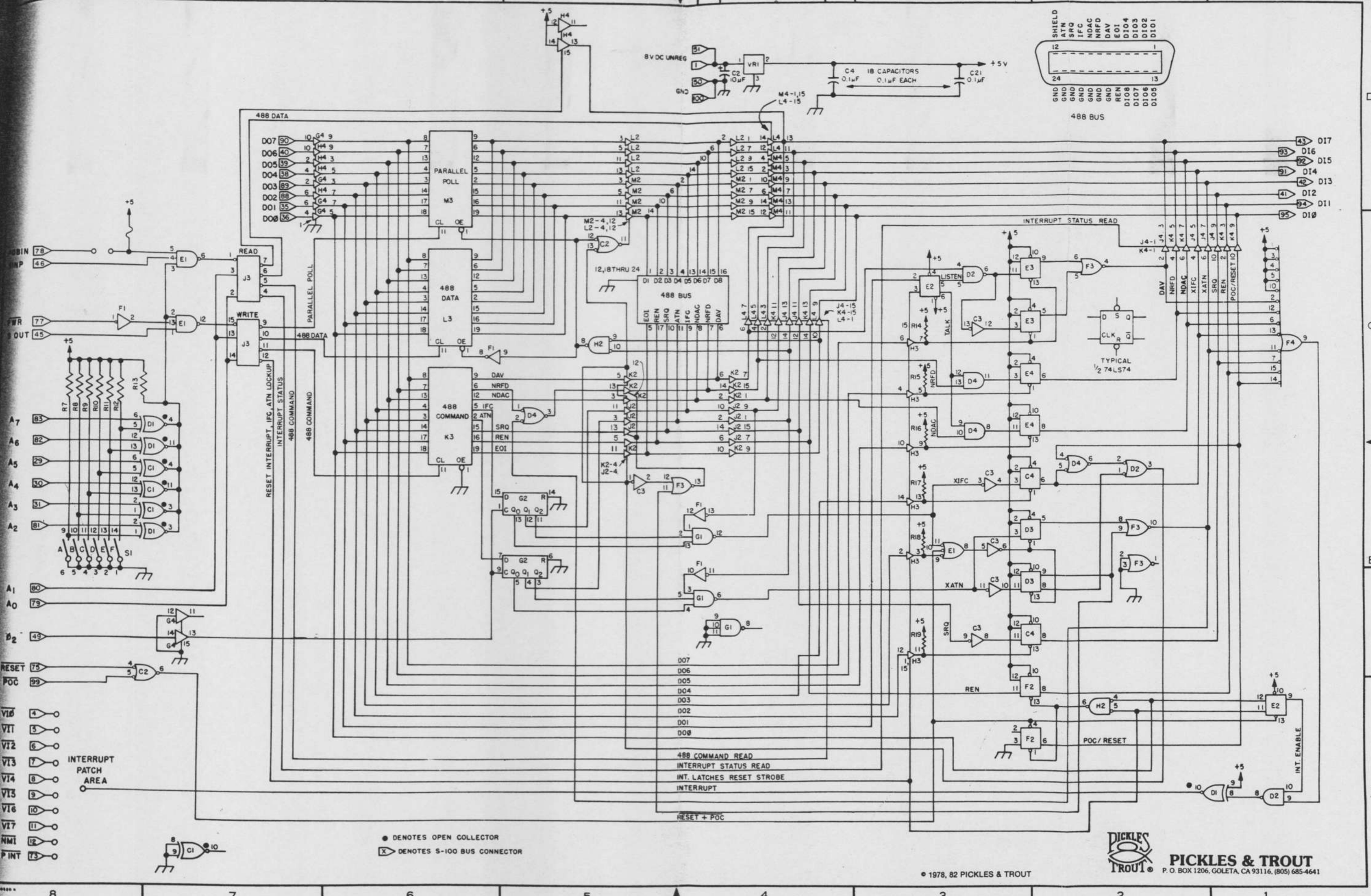
```
        CALL    COMND
        MOV     A,D     ;put the data back in register A
        RET
;
;       Initialize 488 board
;
;  This routine should be called after every S-100 RESET or
;  Power On Clear
;
INIT:   MVI     A,0FFH
        OUT     PPORT   ;clear parallel poll response port
        OUT     DATPT   ;  and 488 data port
        CALL    COMND   ;  and 488 control lines and image byte
        SUB     A
        OUT     ISRPT   ;clear Interrupt Service Register
        STA     RETCOD  ;  clear return code
        STA     CHAR    ;  and CHAR
        RET
;
;  COMND keeps track of the last byte that was output to the
;  command port.  It is necessary to keep track of what the
;  P&T-488 interface board is asserting on the bus because
;  the 488 bus is an open-collector wire-or system, so it is
;  not possible to determine what the P&T-488 is asserting
;  on the 488 bus by merely sensing the 488 lines.
;
COMND:  STA     GIMTC   ;update the 488 command line image
        OUT     CMDPT   ;put it on the command lines
        RET
;
;       Check for interrupt due to ATN, IFC or POC
;
;  NOTE:  This function does not reset the interrupts in the
;         Interrupt Service Register (ISR)
;
INTRPT: IN      ISRPT   ;look at the interrupt service register
        RAR             ;put POC bit in carry
        CNC     IPOC    ;..set POC bit in return code byte if
                        ;  no carry
        RAR             ;REN  > CARRY
        RAR             ;SRQ  > CARRY
        RAR             ;ATN  > CARRY
        CNC     IATN    ;..set the XATN bit
        RAR             ;IFC  > CARRY
        CNC     IIFC    ;..set the XIFC bit
        LDA     RETCOD
        ANI     0F0H    ;look at only POC, IFC and ATN
        RET
;
IPOC:   PUSH    A
        LDA     RETCOD
        ORI     80H
ICOM:   STA     RETCOD
        POP     A       ;restore reg A and carry
```

```
        RET
;
IATN:   PUSH    A
        LDA     RETCOD
        ORI     20H
        JMP     ICOM
;
IIFC:   PUSH    A
        LDA     RETCOD
        ORI     40H
        JMP     ICOM
;
;       Print the reason for aborting then jump to the monitor
;
BYE:    PUSH    PSW     ;save the error code
        LXI     D,MS2   ;power on clear
        ANI     80H
        CNZ     PRINT
        POP     PSW     ;get the error code again
        PUSH    PSW
        LXI     D,MS3   ;XIFC
        ANI     40H
        CNZ     PRINT
        POP     PSW
        LXI     D,MS4   ;XATN
        ANI     20H
        CNZ     PRINT
        JMP     MONITR
;
;       No listeners present - print error message then
;       jump to the monitor
;
NOLSN:  LXI     D,MS1   ;print no listener msg
;
;       Print error message and return to monitor
;
ERROR:  CALL    PRINT
        JMP     MONITR
;
;       print the line pointed to by DE
;
PRINT:  MVI     C,BUFPRN
        CALL    CPMIO
        RET
;
;
GIMTC:  DB      0       ;image of last byte sent to CMDPT
CHAR:   DB      0
RETCOD: DB      0       ;a byte containing the error code

MS1:    DB      'No listeners on the bus',CR,LF,ES
MS2:    DB      'S-100 POWER ON CLEAR or RESET',CR,LF,ES
MS3:    DB      'Another 488 Controller is asserting IFC true',CR,LF,ES
MS4:    DB      'Another 488 Controller is asserting ATN true',CR,LF,ES
```

```
;*****************************************************************
;
;       SAMPLHS.ASM
;
;  This program uses the Acceptor handsahke routine to get a
;  data byte from the IEEE-488 bus and display it on the
;  system console.
;
;*****************************************************************
;
        ORG     100H

MONITR  SET     0       ;CP/M warmstart entry point
CPMIO   SET     5       ;CP/M I/O routine entry point

GETCHR  SET     1       ;CP/M function code for console input
PUTCHR  SET     2       ;CP/M function code for console output
CONSTAT SET     11      ;CP/M function code for console status

        LXI     SP,2000H        ;initialize stack pointer
        CALL    INIT            ;initialize the P&T-488 card
LOOP:   CALL    ACEPTR          ;get a byte from the 488 bus
        MOV     E,A             ;put it in register E for CP/M
        MVI     C,PUTCHR        ;function to print on console
        CALL    CPMIO           ;CP/M I/O routine entry point
        MVI     C,CONSTAT       ;look to see if a key is pressed
        CALL    CPMIO
        ANI     1
        JZ      LOOP            ;..no key pressed
        MVI     C,GETCHR        ;get the key
        CALL    CPMIO
        CPI     3               ;CONTROL C?
        JNZ     LOOP            ;..no, so continue getting data
                                ;  from the bus
        JMP     MONITR          ;..yes, so do a warmstart

;*****************************************************************
;
;       Insert the Handshake routines here
;
;*****************************************************************

        END
```



P & T 488 REV. 81A


8V DC UNREG
GND
VR1
C2 10µF
C4 0.1µF 18 CAPACITORS 0.1µF EACH
C21 0.1µF
+5V
SHIELD ATN SRQ IFC NDAC NRFD DAV EOI DIO4 DIO3 DIO2 DIO1
GND GND GND GND GND GND GND REN DIO8 DIO7 DIO6 DIO5
488 BUS
488 DATA
DO7 DO6 DO5 DO4 DO3 DO2 DO1 DO0
PARALLEL POLL M3
488 DATA L3
488 COMMAND K3
M2-4,12 L2-4,12
12,18THRU 24
D1 D2 D3 D4 D5 D6 D7 D8
488 BUS
EOI REN SRQ ATN IFC NDAC NRFD DAV
J4-15 K4-15 L4-1
INTERRUPT STATUS READ
DI7 DI6 DI5 DI4 DI3 DI2 DI1 DI0
DAV NRFD NDAC XIFC XATN SRQ REN POC/RESET
TYPICAL 1/2 74LS74
LISTEN
TALK
NRFD
NDAC
XIFC
XATN
SRQ
REN
POC/RESET
INT. ENABLE
READ
WRITE
PARALLEL POLL
488 DATA
INTERRUPT, IFC, ATN LOCKUP
INTERRUPT STATUS
488 COMMAND
488 COMMAND
RESET INTERRUPT
DAV
NRFD
NDAC
IFC
ATN
SRQ
REN
EOI
A7 A6 A5 A4 A3 A2 A1 A0 D2
RESET POC
VI0 VI1 VI2 VI3 VI4 VI5 VI6 VI7 NMI PINT
INTERRUPT PATCH AREA
DO7 DO6 DO5 DO4 DO3 DO2 DO1 DO0
488 COMMAND READ
INTERRUPT STATUS READ
INT. LATCHES RESET STROBE
INTERRUPT
RESET + POC
● DENOTES OPEN COLLECTOR
X DENOTES S-100 BUS CONNECTOR
© 1978, 82 PICKLES & TROUT
PICKLES & TROUT
P. O. BOX 1206, GOLETA, CA 93116, (805) 685-4641