zSBX 20

GPIB Controller Interface
Rev A or B Boards

REORDER PART NUMBER ZT MzSBX 20
September 14, 1989

Ziatech Corporation
3433 Roberto Court
San Luis Obispo, CA 93401
ITT Telex 4992316
FAX (805) 541-5088
Telephone (805) 541-0488

INTRODUCTION

The SBX Bus is a unique interface facilitating on-board I/O expansion with SBX Multimodule boards. The SBX Bus is driven directly by the host board, thus becoming an integral element of a computer's I/O system. The host board with SBX Bus capability brings a new concept to I/O expansion, enabling users to tailor their applications directly at a minimal cost with a savings in I/O slots. Please refer to Figure 1, SBX Multimodule Board Concept.

At present, SBX Multimodule boards are available from numerous manufacturers. Some twenty different functions are supported.

This manual covers the Ziatech zSBX 20 IEEE 488 (GPIB, HPIB) Multimodule interface board. Both hardware and software descriptions will be covered in addition to an IEEE 488 tutorial.

zSBX 20 FEATURES

* IEEE 488-1978 Standardized computer I/O
* Complete controller, talker, and listener capability
* Standard peripheral and instrument interface
* DMA compatible for performance enhancement
* Easily programmed for fast system development

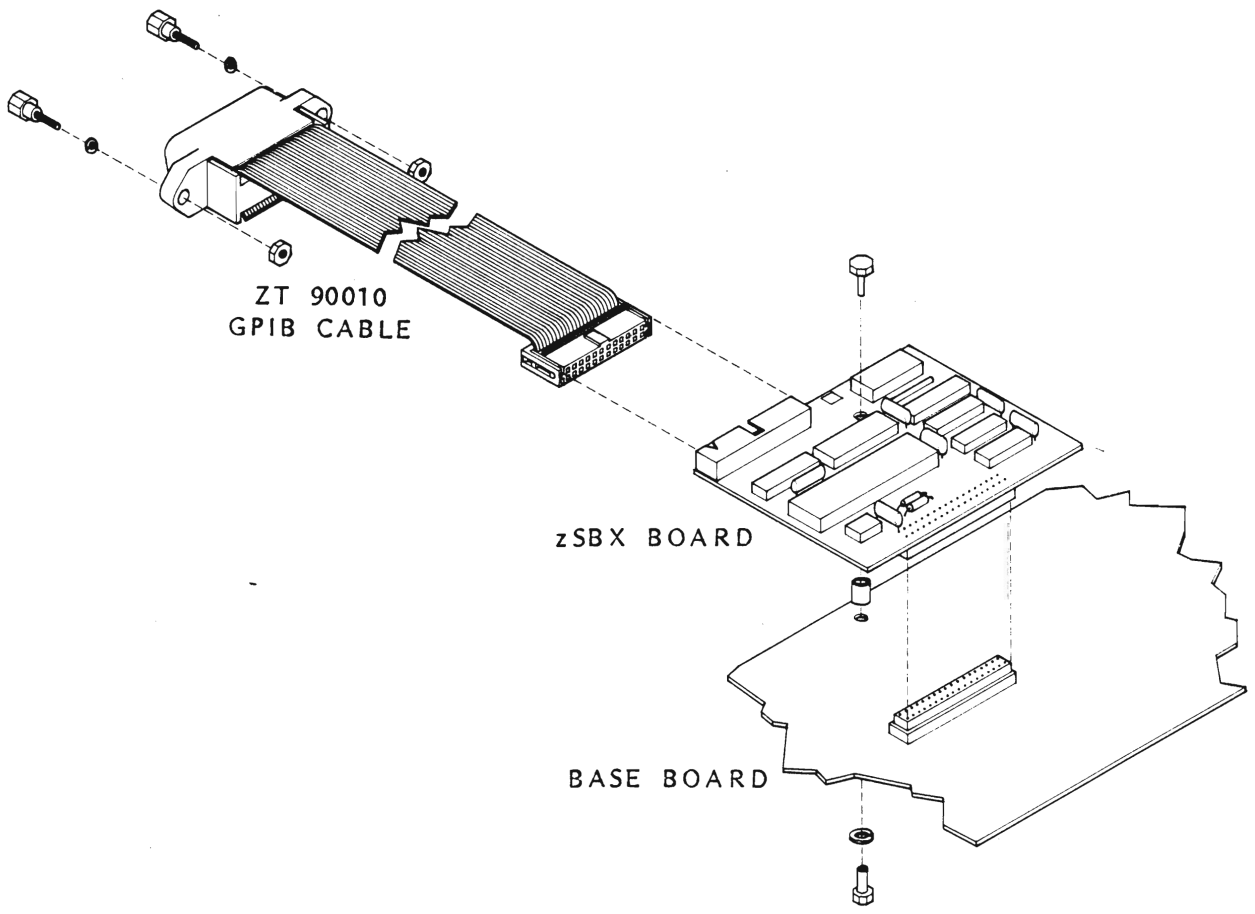


Figure 1. SBX Multimodule Board Concept.

SYSTEM DESCRIPTION

The zSBX 20 Multimodule board is a small, I/O mapped, IEEE 488 interface board which plugs into base boards. The SBX board connects to the SBX base board connector via a standard unique connector designed specifically for the SBX Bus. The SBX Bus on the base board is brought out to a female connector, while the SBX Multimodule board mates with a male connector (Figure 2, zSBX 20 IEEE 488 Multimodule Installation).

The Multimodule board and base board are both mechanically connected together in two places. These two points are the SBX connector and a nylon screw/spacer assembly that are provided. The screw is a 6-32 x 1/4 inch long, and the spacer is a 6-32 threaded x 1/2 inch long.

The zSBX 20 Multimodule IEEE 488 interface board is provided with a 14 inch GPIB ribbon cable. This cable connects to the zSBX 20 Multimodule board and is terminated with an IEEE 488 GPIB female panel-mount connector. Connector jack screws conforming to the IEEE 488 Standard are also provided.

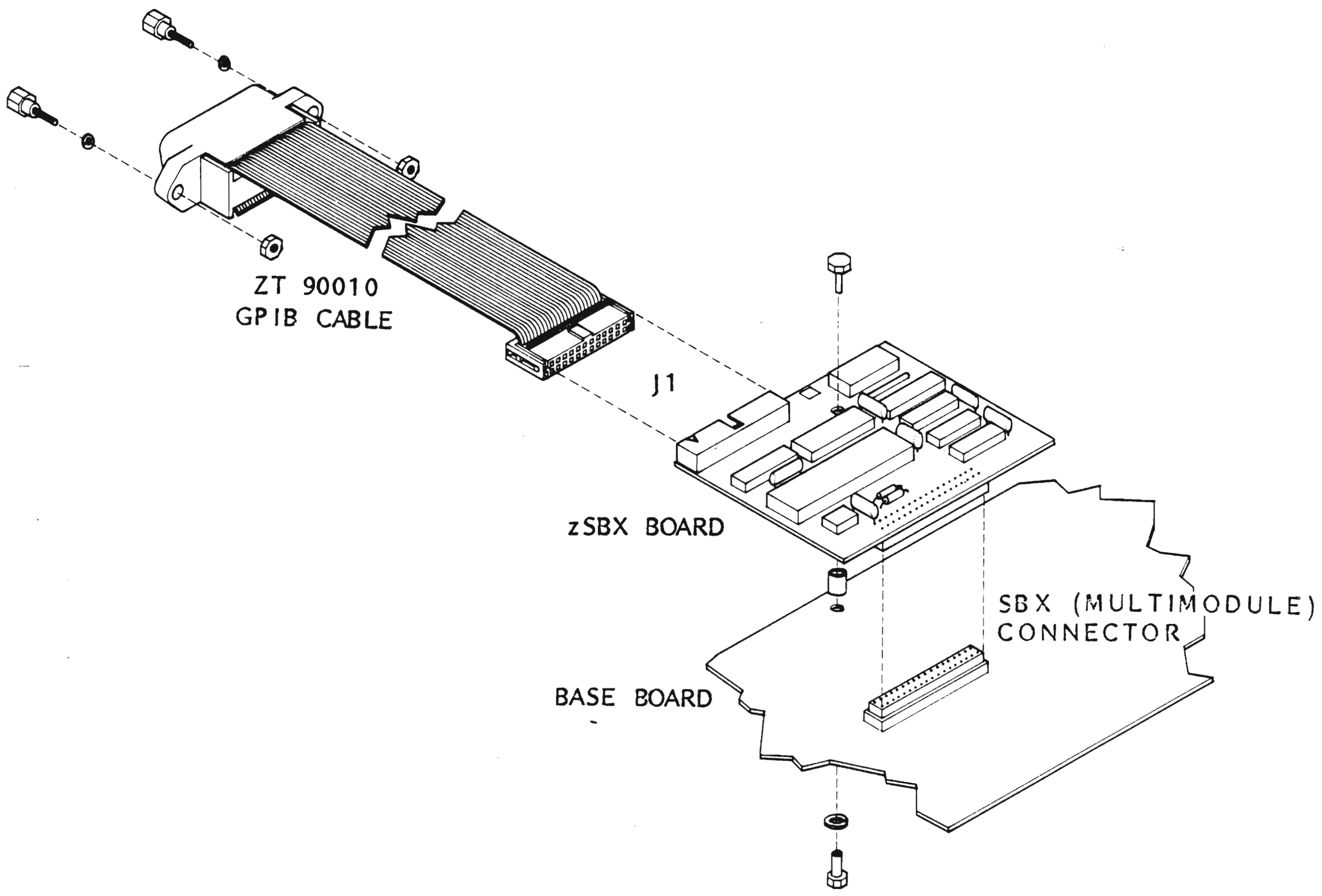


Figure 2. zSBX 20 IEEE 488 Multimodule Installation.

zSBX 20 THEORY OF OPERATION

The Ziatech zSBX 20 is a self-contained IEEE 488 GPIB (General Purpose Interface Bus) interface for the SBX Bus. It is electrically, physically, and logically compatible with the SBX Bus. Figure 3 illustrates the basic zSBX 20 functional diagram.

The SBX Interface supports both DMA and polling operations by SBX base boards. For DMA operation, the base board must support DMA operation by using MDRQT and MDACK signal lines to transfer data. If DMA is not supported, polling must be used to transfer data. Polling operations must be used to initialize the GPIB section and read the GPIB address switch.

The GPIB section provides the protocol to conform to the IEEE 488 - 1978 Standard. Complete talker, listener and controller capabilities are provided. Primary and secondary address capability is provided. Interrupts can also be generated on various GPIB conditions.

The GPIB address switch section provides access to an 8-position DIP switch. The GPIB device address is set by 5 switches. This DIP switch can also be remotely mounted (for example, on a back panel) by using Ziatech's ZT 90007 remote DIP switch cable.

The transceiver section interfaces to the IEEE 488 lines to provide electrical characteristics conforming to the IEEE 488 - 1978 Standard Specification. Open-collector or three-state characteristics are available.

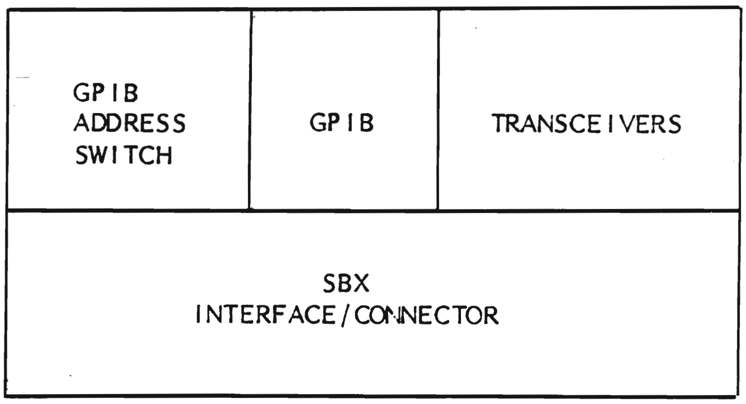


Figure 3. zSBX 20 Functional Diagram.

zSBX 20 APPLICATION EXAMPLES

EXAMPLE #1, USING THE zSBX 20 TO CONSTRUCT A TEST SYSTEM

OBJECTIVES

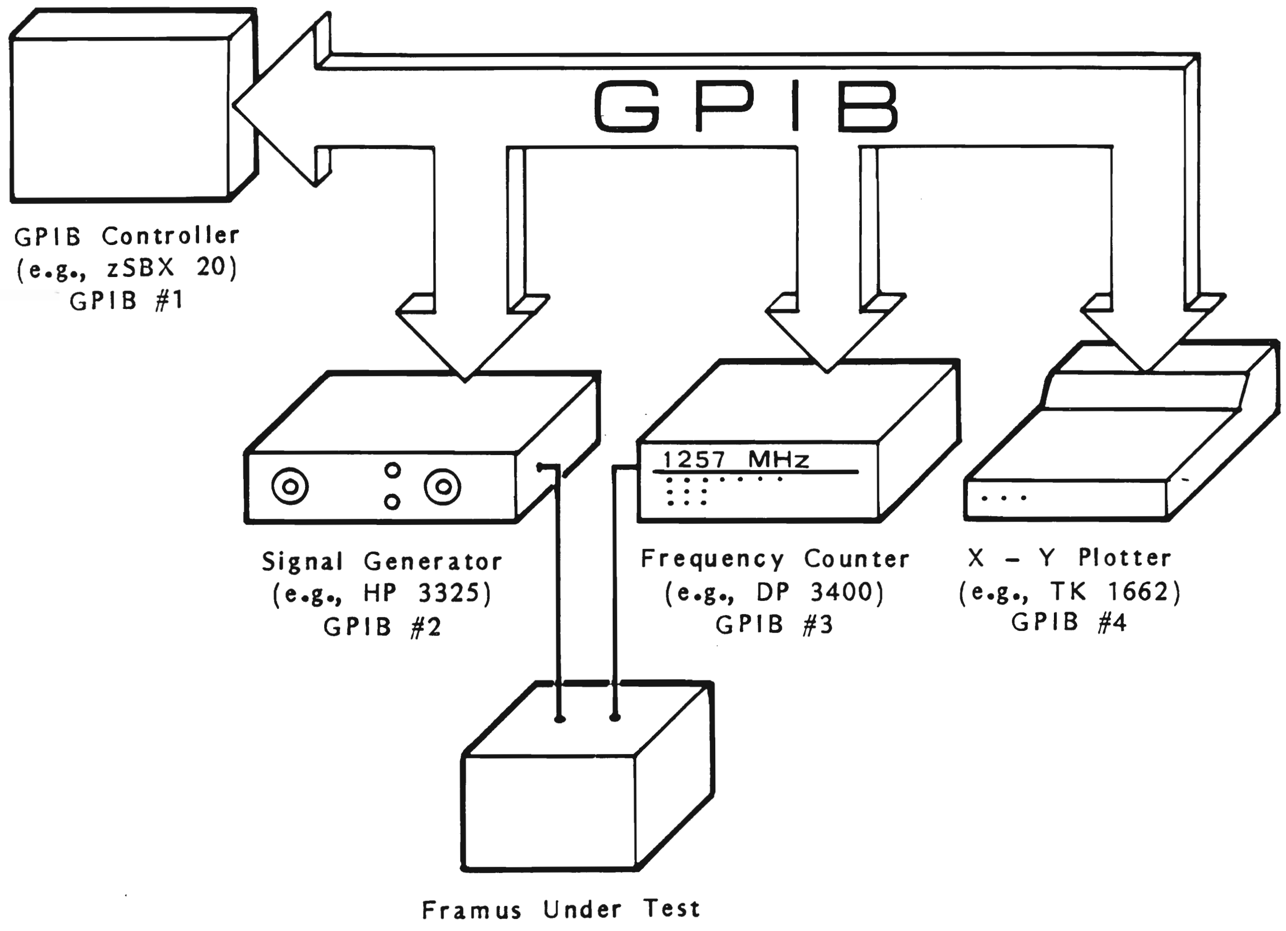
* Develop an automated test equipment (ATE) station to inject various stimulus signals into a unit under test (UUT) in order to measure and record the output results using the GPIB

* Illustrate message protocol in a (>40kHz) GPIB system configured with the following:

  - zSBX 20 GPIB controller (talker, listener, controller)
  - Signal generator (listener)
  - Frequency counter (talker, listener)
  - Plotter (talker, listener)

* Illustrate the zSBX 20 software drivers executed by the 8085 CPU (Note: 8086 drivers are available also. See Appendix C)

BLOCK DIAGRAM



SOFTWARE DESCRIPTION

* Configure zSBX 20 as system controller by calling the Initialization routine
* Put GPIB into Remote
* Configure GPIB devices 2 & 3 by addressing each to listen and sending the appropriate device commands via the Send routine
* Start the signal generator and frequency counter via the Send routine
* Read the frequency counter reading into the MULTIBUS† system via the Receive routine
* Process and scale reading for output to the plotter
* Output plot data to plotter via the Send routine
* Repeat all but first two steps for each unit under test

SOFTWARE EXAMPLE

* Assuming an 8080 or 8085 CPU is used to configure the zSBX 20, a call as follows would be made:

```
            CALL     INIT      ; Initialize GPIB
            CALL     INT17     ; Initialize DMA
            CALL     REME      ; Remote Enable
```

* Configure the HP3325A function generator for a sine wave of 5kHz with an amplitude of 3 Vp-p and an offset of 1.5V:

```
            LXI      H,LSTNL2  ; Load listen list address
            LXI      D,BUFF2   ; Load buffer address
            LXI      B,BUFFL2  ; Load buffer length
            CALL     SEND      ; Configure HP3325
             .
             .
             .
  LSTNL2:   DB       22H       ; HP3325 listen GPIB addr #2
            DB       FFH       ; Listener list end
  BUFF2:    DS       'FU1FR5KHAM3VOOF1.5VO
                                          Offset
                                          Amplitude
                                          Frequency
                                          Function

  BUFFL2:   EQU      $-BUFF2   ; Length of message
```

* Configure the Data Precision 3400 DMM with the 3410 GPIB adapter for an AC measurement on the 10V range using an immediate trigger mode to initialize each measurement and then request service:

†MULTIBUS is a trademark of Intel Corporation

```
                LXI     H,LSTNL3   ; Load listen list address
                LXI     D,BUFF3    ; Load buffer address
                LXI     B,BUFFL3   ; Load buffer length
                CALL    SEND       ; Configure DP3400
                 .
                 .
                 .
        LSTNL3: DB      23H        ; DP3400 listen GPIB addr #3
                DB      FFH        ; Listener list end
        BUFF3:  DS      'F2R7T1M5'
                         |  |  |  |
                         |  |  |  +------ SRQ w/each measurement
                         |  |  +--------- immediate trigger
                         |  +------------ autorange
                         +--------------- AC Volts

        BUFF3L  EQU     $-BUFF3
```

* To make a measurement, send execute message:

```
        START:  LXI     H,LSTNL3   ; Load listener list
                LXI     D,BUFF7    ; Load buffer address
                LXI     B,BUFFL7   ; Load buffer length
                 .
                 .
                 .

        BUFF7:  DS      'E'        ; Start reading
        BUFFL7  EQU     $-BUFF7
```

* To wait for a Service Request (SRQ) from the DMM indicating a completed measurement, call the Service Requested routine. Note: this could be done with interrupts if the processor is needed for other things.

```
        LOOP:   CALL    SRQ        ; Wait for SRQ
                JZ      LOOP
                 .
                 .
                 .
```

* When a service request is pending, do a serial poll and get the response byte:

```
                LXI     H,TLKSP    ; Serial poll talker list
                LXI     D,RESP     ; SP response byte buffer
                CALL    SPOL       ; Do a serial poll
                 .
                 .
                 .

        TLKSP:  DB      43H        ; DP3400 talker GPIB addr
                DB      FFH        ; Talker list end
        RESP:   DS      1          ; Storage for response byte
```

* Check the response byte to see if the DP3400 DMM has a reading:

```
                LXI     H,RESP     ; Get address of response
                MOV     A,M        ; Get response byte
                ANI     40H        ; Mask for positive response
                JZ      OTHER      ; If not, do other processing
                 .                 ; Yes, needs service
                 .
                 .
```

* Get DMM reading:

```
                LXI     H,TLKRL3   ; Load talker list address
                LXI     D,BUFF4    ; Load buffer address
                LXI     B,BUFFL4   ; Load buffer length
                CALL    RECV       ; Get measurement
                 .
                 .
                 .

       TLKRL3:  DB      43H        ; DP3400 talker GPIB addr #3
                DB      FFH        ; Talker list end
       BUFF4:   DS      4          ; Storage for measurement
       BUFF4L   EQU     $-BUFF4    ; Length of measurement
```

* MULTIBUS system now processes and scales data for output to the plotter:

```
                LXI     H,BUFF4    ; Get measurement address
                MOV     A,M        ; Get data
                 .                 ; Process & scale
                 .
                 .
                LXI     H,BUFF6    ; Get output address
                MOV     M,A        ; Store data
```

* Data to be sent to the plotter is output via the Send routine:

```
                LXI     H,LSTNL4   ; Load listener list address
                LXI     D,BUFF5    ; Load buffer address
                LXI     B,BUFF5L   ; Load buffer length
                 .
                 .
                 .

       LSTNL4:  DB      24H        ; TEK4662 listener GPIB addr
                DB      FFH        ; Talker list end
       BUFF5:   DB      'SP'
                         ||
                         |└──────────── print ASCII string
                         └───────────── alpha mode print

       BUFF6:   DS      4          ; Storage for measurement
       BUFF5L   EQU     $-BUFF5
```

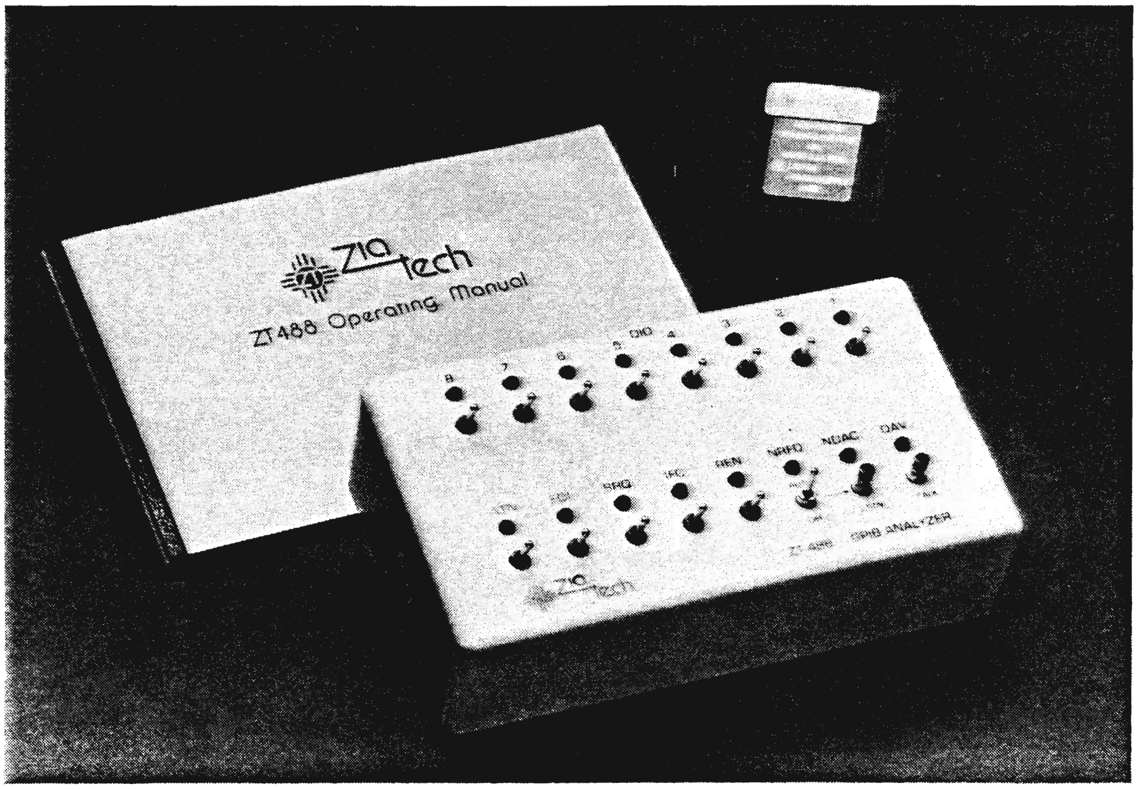
* To get another reading, jump back to START to initiate another measurement:

```
                JMP     START
```

OTHER CONSIDERATIONS

* This example could be made more efficient by increasing the buffer size for the plotter before printing, thus decreasing the addressing overhead. Note also that the majority of GPIB systems will use the service request line to determine the status of the GPIB devices and subsequently service them. The CPU idle time could be reduced by interrupting upon SRQ instead of polling.

* To facilitate efficient GPIB debugging a GPIB analyzer would be useful. One such analyzer is manufactured by Ziatech (ZT 488), allowing bus monitoring while stepping through bus transactions one at a time. When utilized in the above system, the analyzer is used to break apart the system, isolating each component for operational verification. For example, the sequence used to initialize the function generator could be sent by the analyzer to check for proper operation. As an alternative, the analyzer could be connected to the output of the zSBX 20 to verify proper programming. When each component is verified with the analyzer, all components can now be connected together with the analyzer included to verify proper system operation. Whether the analyzer is used as a device or controller, the unit usually pays for itself during the first week of debugging a GPIB system.



ZT 488 GPIB Analyzer with Operator's Manual
and 115V AC Power Supply.

EXAMPLE #2, USING THE zSBX 20 AS A FRONT-END PROCESSOR

OBJECTIVES

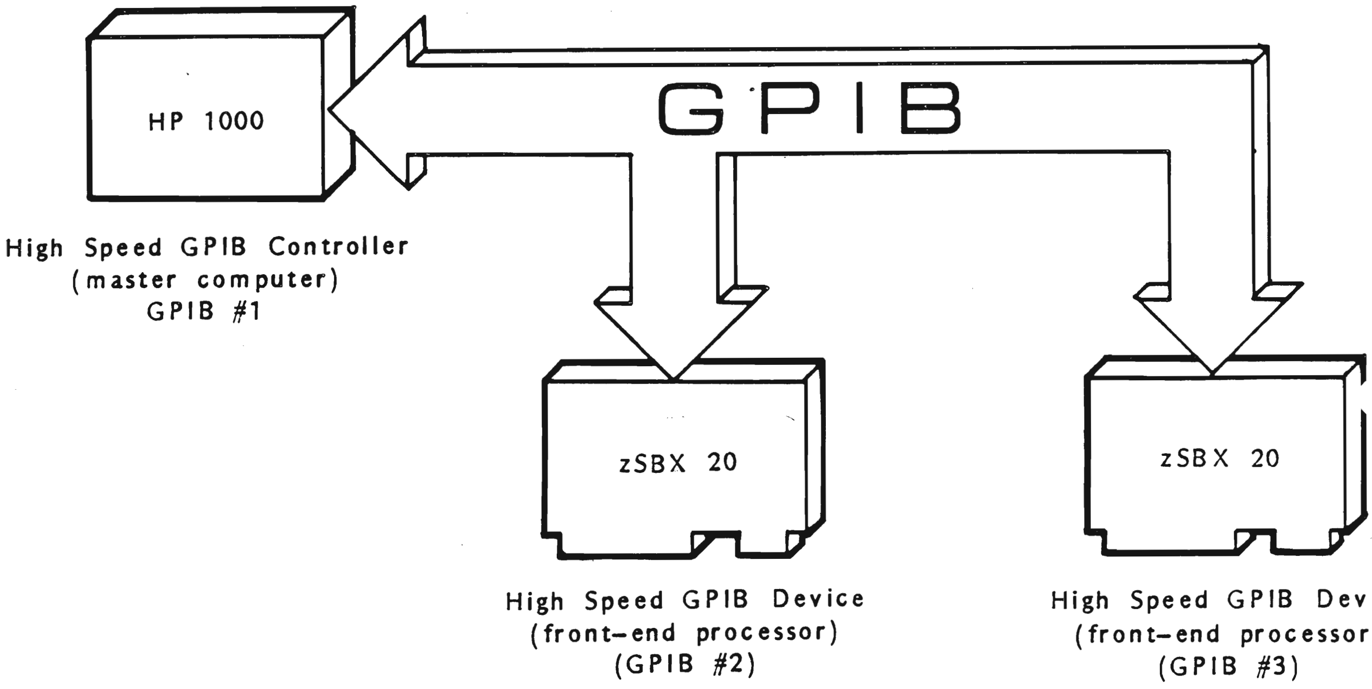
* Develop a high speed (<250kHz) computer-to-computer GPIB communications in a system utilizing front-end processors

* Illustrate message protocol in the GPIB system configured with the following:

  - GPIB Controller (talker, listener, controller)
  - zSBX 20 GPIB Device (talker, listener)
  - zSBX 20 GPIB Device (talker, listener)

* Illustrate the zSBX 20 software drivers executed by the 8085 CPU's

BLOCK DIAGRAM


HP 1000
GPIB
zSBX 20
zSBX 20
High Speed GPIB Controller (master computer) GPIB #1
High Speed GPIB Device (front-end processor) (GPIB #2)
High Speed GPIB Device (front-end processor) (GPIB #3)

OTHER CONSIDERATIONS

* The GPIB provides an easy extension to more than 2 front-end processors. The limit will be determined by the quantity of data to be transferred and the required number of transactions per second. The GPIB places an upper limit of 15 devices in the system.

* The front-end processors used in this example must be local to the mainframe because the GPIB has a length restriction of less than 2 meters per device. For small numbers of devices as shown here, the bus may be preloaded with equivalent loads to simulate 10 devices, allowing the GPIB maximum length of 20 meters.

* For high-speed communications, the GPIB is faster than most RS-232C links.

* The GPIB makes a good system-independent communication protocol for high-speed, local systems. Most mini-computers and many mainframe computers have GPIB controllers and there are over 1000 GPIB compatible instruments available today for system use.

EXAMPLE #3, USING THE zSBX 20 WITH IBM PC BASIC

OBJECTIVES

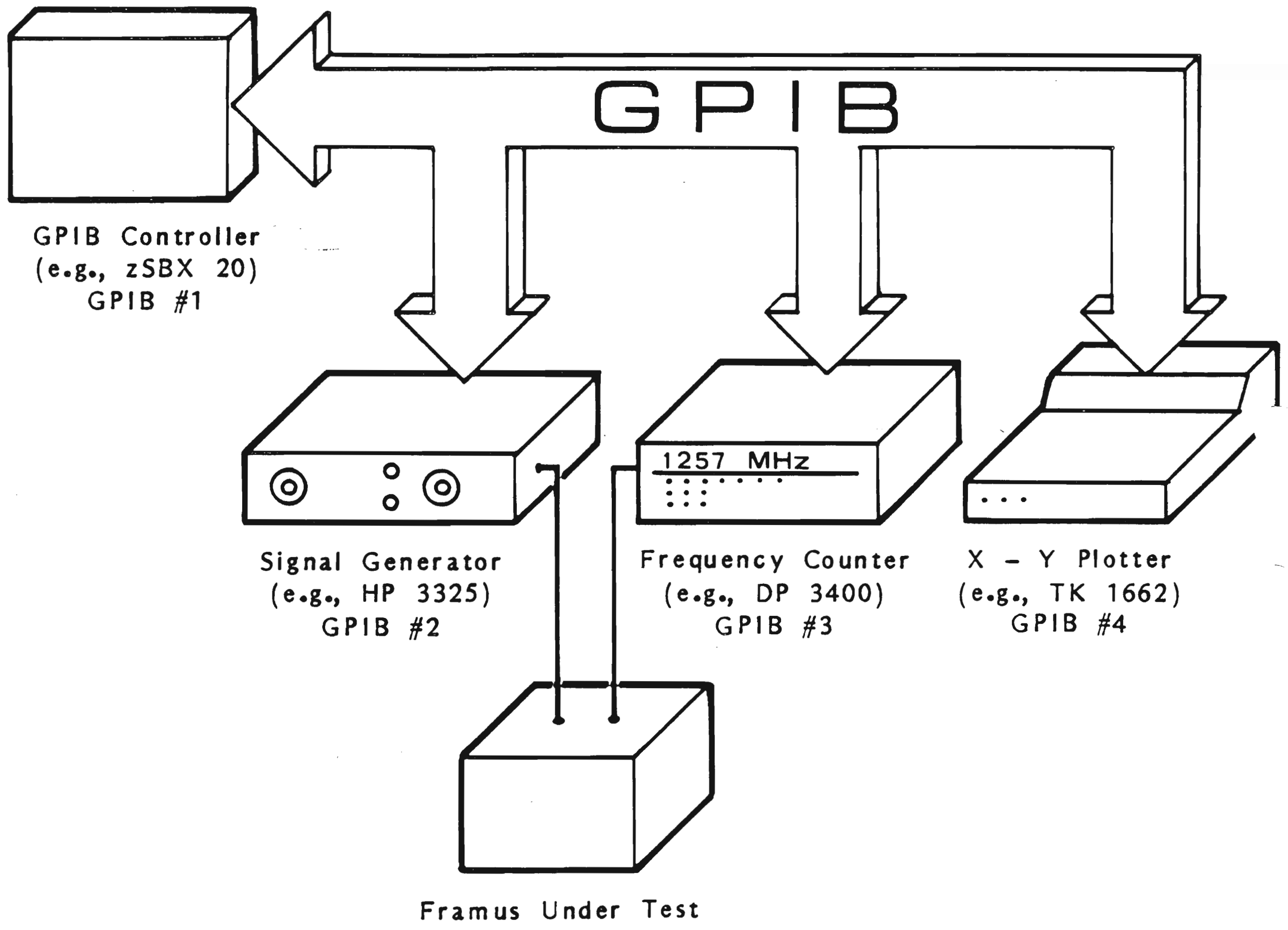
* Develop an automated test equipment (ATE) station to inject various stimulus signals into a unit under test (UUT) in order to measure and record the output results using the GPIB

* Illustrate message protocol in a (>40kHz) GPIB system configured with the following:

  - zSBX 20 GPIB controller (talker, listener, controller)
  - Signal generator (listener)
  - Frequency counter (talker, listener)
  - Plotter (talker, listener)

* Illustrate the zSBX 20 with the IBM PC.
  (Note: 8086 drivers are available also. See Appendix C)

BLOCK DIAGRAM



SOFTWARE DESCRIPTION

* Configure zSBX 20 as system controller by calling the Initialization routine
* Put GPIB into Remote
* Configure GPIB devices 2 & 3 by addressing each to listen and sending the appropriate device commands via the Send routine
* Start the signal generator and frequency counter via the Send routine
* Read the frequency counter reading into the IBM PC system via the Receive routine
* Process and scale reading for output to the plotter
* Output plot data to plotter via the Send routine
* Repeat all but first two steps for each unit under test

SOFTWARE EXAMPLE

* Assuming an IBM Personal Computer is used to configure the zSBX 20, a call as follows would be made:

```
100     CALL INIT                ' Initialize GPIB
```

* Configure the HP3325A function generator for a sine wave of 5kHz with an amplitude of 3 Vp-p and an offset of 1.5V:

```
110     FGEN$ = "2 "             ' HP3325 GPIB addr #2
120     CONFIG$ = "FU1FR5KHAM3VOOF1.5VO"
                                          Offset
                                          Amplitude
                                          Frequency
                                          Function
130     CALL SENDS (FGEN$,CONFIG$)
```

* Configure the Data Precision 3400 DMM with the 3410 GPIB adapter for an AC measurement on the 10V range using an immediate trigger mode to initialize each measurement and then request service:

```
140     DVM$ = "3 "              ' DP 3400 GPIB addr #3
150     CONFIG$ = "F2R7T1M5"
                                 SRQ w/each measurement
                                 Immediate trigger
                                 autorange
                                 AC Volts
160     CALL SENDS (DVM$,CONFIG$)
```

* To make a measurement, send execute message:

```
170     EXE$ = "E"
180     CALL SENDS (DVM$,EXE$)
```

* To wait for a Service Request (SRQ) from the DMM indicating a completed measurement, call the Service Requested routine.

```
190     CALL SRQD (SERVICE%)     ' Wait for SRQ
200     IF SERVICE%<>0 THEN GOTO 205
           ELSE GOTO 190
```

* When a service request is pending, do a serial poll and get the response byte:

```
205     STATUS$ = SPACE$(1)
210     DEVICE$ = SPACE$(3)
215     CALL SPL (DVM$,STATUS$,DEVICE$)
```

* Check the response byte to see if the DP3400 DMM has a reading:

```
220     IF VAL(STATUS$)AND(40) THEN GOTO 225
           ELSE GOTO ?
```

* Get DMM reading:

```
225    DATUM$ = SPACE$(20)
230     CALL RECVS (DVM$,DATUM$)
```

* IBM PC now processes and scales data for output to the plotter:

```
                •
                •
                •
```

* Data to be sent to the plotter is output via the Send routine:

```
240     PLTR$ = "4 "            ' TEK4662 GPIB ADDR
250     CALL SENDS (PLTR$,GRAPH$)
```

* To get another reading, jump back to initiate another measurement:

```
260     GOTO 180
 •
 •
 •
999     END
```

INTRODUCTION TO THE GPIB CAPABILITIES OF THE zSBX 20

The IEEE 488 Bus (GPIB) is truly a universal interface. Regardless of the device being interfaced, its function or purpose, all devices on the bus look and act the same. The Three-World Interface shown in Figure 4, demonstrates how three completely foreign worlds: the world of microcomputers, the world of instrumentation, and the world of industrial I/O, can all operate together as a single system via a few basic functions.

A GPIB device can only perform three basic functions or subsets of functions. A GPIB device can either talk (send data), listen (receive data), or control (i.e., determine who talks or listens). This is a simplified summary of GPIB activity.

In Figure 4, for example, say that Device 2 wants to talk to Device 3, and assume that Device 1 is the controller. What Device 2 wants to "say" is totally irrelevant to GPIB activity. The "conversation" could be, for example, the latest position readings of a work piece being cut by a robotic milling machine. Device 3 may need the position reading to adjust a set of servo motors to move the work piece to a new position. How does this "conversation" take place?

When Device 2 needs to talk, it can attract the "attention" of the controller, Device 1, by asserting a dedicated service line called "Service Request" (SRQ). When the controller senses SRQ, it knows that one or more of the other devices on the GPIB requested some kind of assistance or "service". The controller can easily determine which device requested service and the type of service requested, if any, by performing one of several "polling" techniques.

GPIB polling will be discussed later, but for now, assume that the controller knows that Device 2 must talk to Device 3 and Device 3 must listen to Device 2. Before devices can communicate with each other, they first must be "introduced" by the controller.

The controller can start the introduction by "addressing" Device 2 to talk. How does the controller do this? All devices will give their undivided "attention" to the controller when the controller is asserting its "Attention"

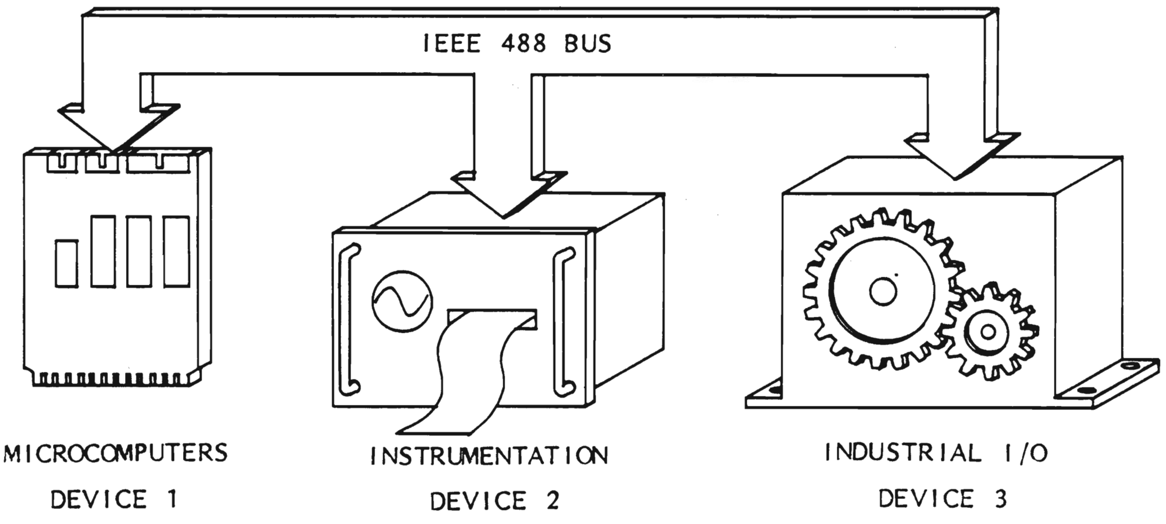


Figure 4. Three-World Interface.

(ATN) line, so when Device 2 sees his own private talk address accompanied with ATN, it knows that its turn to give a "speech" is coming up next.

The controller can remove any possible uninvited listeners or "eavesdroppers" from the bus by sending the Universal Unlisten (UNL) message, accompanied with ATN. When UNL is sent by the controller, all listeners become inactive.

The controller can selectively invite listeners, in this case only Device 3, by sending listen addresses to the GPIB. When Device 3 "sees" its unique listen address asserted on the GPIB with ATN, it knows it soon must listen to all conversations.

Assume that the controller addressed Device 2 to talk and Device 3 to listen. The conversation begins as soon as the controller relinquishes control of the GPIB by removing the ATN signal. This tells all devices participating in the conversation that they may now give their "attention" to each other.

There are several ways to terminate a "conversation" or data transfer. In all cases, the talker should inform the listener(s) that the transfer is complete. To accomplish this, the GPIB has a dedicated line called End-Or-Identify (EOI). When Device 2 is sending the last word, it can assert the EOI line, indicating "End". This is good system design, giving Device 3 or any other listeners, the opportunity to accept the last byte and immediately begin to process the received data.

If the system does not use EOI as an end indicator, the talkers and listeners should be pre-programmed to transfer either a prearranged string length, or agree on a standardized "End-Of-String" (EOS) character. The EOS character can be anything the system designer chooses. Some systems employ an ASCII Carriage Return or Line Feed character. The EOS character may even change dynamically so long as the system is designed to inform all devices of the change.

If Device 2 is finished talking, the controller can take control of the GPIB again by asserting ATN. The controller could take control before Device 2 is finished talking and stop the transfer between words, or even in the middle of a word (destroying the word); however, this is not good system practice. The controller should only take control in the middle of a transfer, in the event of a serious system failure.

When the controller takes charge again, it could set up another conversation with possibly another talker and another set of listeners. The controller could even participate in the conversation by specifying itself as the talker or one of the listeners. The previous talker is usually "untallked", that is, unaddressed by a universal untalk (UNT) or is automatically disabled as talker when another talker other than itself is specified or addressed to talk.

When a device requests service from the controller via the SRQ line, the controller must find out which device or devices requested service, and what kind of service is needed. If the GPIB Serial Poll (SPOL) method is used, the controller responds to the SRQ much like a busy traffic control cop could respond to a tap on the shoulder. The traffic controller would finish conducting the traffic through the intersection and then stop the traffic temporarily. The controller would then interrogate the device that requested service to determine the priority of the service. If it is not "rush hour" the service could be performed immediately, or if the service is a higher priority than the current traffic situation, this would also warrant immediate attention. If there is more than one device on the GPIB capable of requesting service, the controller can sequentially or "serially" poll each device for interrogation.

A controller begins a serial poll by sending the Serial Poll Enable (SPE) message to the GPIB, with ATN asserted. All devices will then expect to be eventually polled. To poll a device, the controller addresses that device to talk. When the controller releases ATN the polled device responds by putting a "Serial Poll Response Byte" on the GPIB. The polled device gets only one "vote": "yes", I did assert SRQ, or "no", I did not.

The vote or "response" bit has been assigned by the GPIB standard to be bit number 7, and it is asserted true if the polled device requested service. The remaining seven bits can contain status information, such as the type of service needed, if any. The remaining status is optional and is left open for the system designer to decide on. When all devices have been polled, the controller sends the Serial Poll Disable (SPD) messsage, telling all devices that the serial poll process is complete.

There is another method of polling called parallel poll. Rather than using bit 7 to indicate an affirmative service requirement, each device on the GPIB has a dedicated data line to assert when polled. With parallel poll, the controller polls all devices at once in "parallel" fashion rather than serially. The polled devices respond by driving their dedicated response line true if service is required, false if not.

In the simplest form of parallel poll, the controller asserts the EOI line with ATN, telling all devices to "identify" by driving their pre-assigned parallel poll response line. There are more ramifications of the parallel poll technique that will not be discussed here. Let it suffice to say that parallel poll is a powerful polling technique that requires careful system design.

THE GPIB ADAPTER (TMS 9914A)

The zSBX 20 implements the full IEEE 488 bus (GPIB) interface using the Texas Instrument TMS 9914A GPIB adapter. A block diagram of the TMS 9914A is given in Figure 5. It is used to interface between the IEEE 488 - 1978 General Purpose Interface Bus

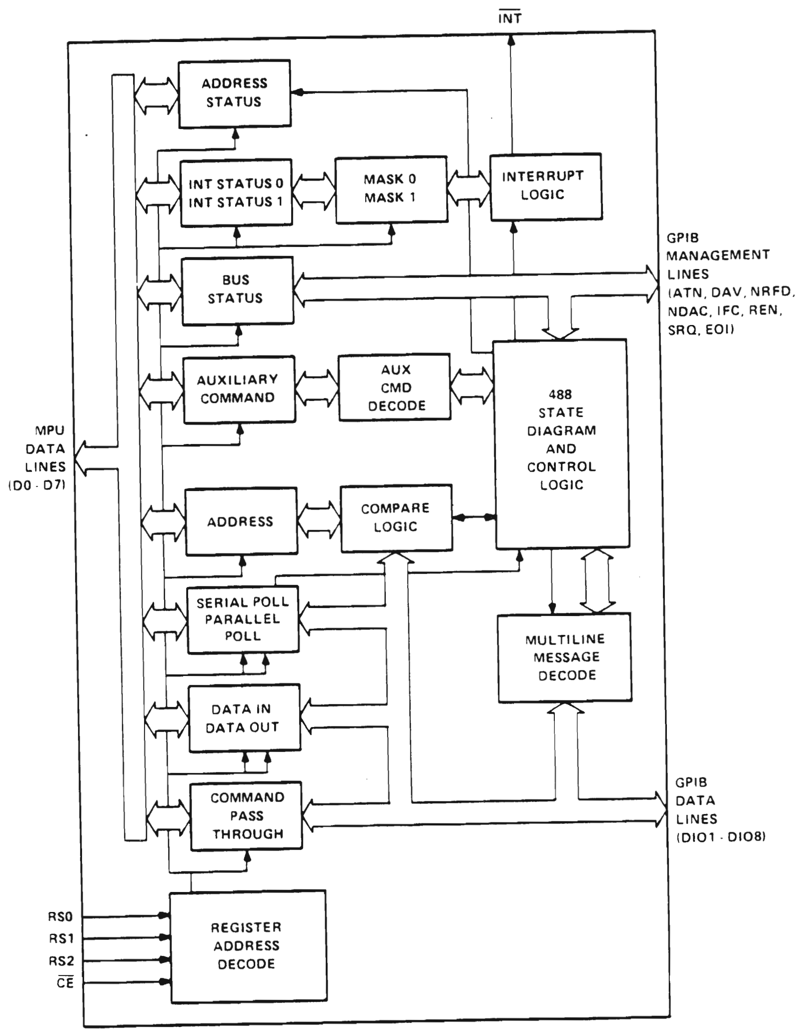


Figure 5. TMS 9914A Block Diagram.

(GPIB) and the off-board CPU. The TMS 9914A interfaces with the GPIB via IEEE 488 transceivers. The TMS 9914A is mapped into the CPU I/O system. The TMS 9914A has 13 accessible registers, seven write and six read. All communication between the GPIB and the microprocessor is carried out via these registers. A summary of each register appears in Figure 6.

I/O PORT ADDR,BASE+	I/O READ REGISTER	I/O WRITE REGISTER
0H	INT STATUS 0	INT MASK 0
1H	INT STATUS 1	INT MASK 1
2H	ADD STATUS	—
3H	BUS STATUS	AUX CMD
4H	ADD SWITCH	ADD REG
5H	—	SER POLL
6H	CMD PASS THR	PAR POLL
7H	DATA IN	DATA OUT

Figure 6. I/O Port Descriptions For The TMS 9914A.

ADDRESS REGISTER (BASE+4H, W) TALKER/LISTENER

The Address Register (ADDR) is a write-only register at base+4H that is written when the zSBX 20 is used as a GPIB Talker/Listener and not a controller. The Address Register engages three major functions of the zSBX 20 as a GPIB Talker/Listener. The following three paragraphs describe these three functions. Consult the TMS 9914A Address Register figure for reference.

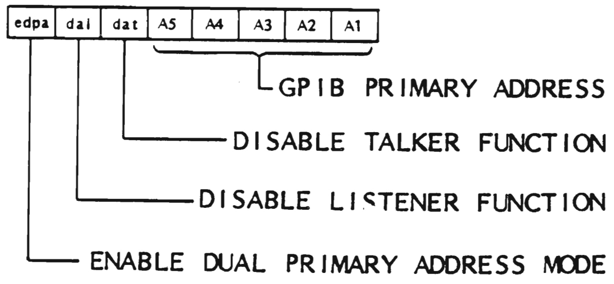


Figure 7. TMS 9914A Address Register.

Every GPIB device requires a five bit address, A5–A1, to distinguish it from other GPIB devices. The Address Register bits A5–A1, when written to, establishes the zSBX 20 five bit GPIB address. The user is advised to get the address by reading the on-board DIP switch at the Address Switch Register, also at port base+4H (see Address Switch Register description below). The address 11111B is not allowed by the IEEE 488 Standard.

The disable-talker (dat) and disable-listener (dal) bits configure the zSBX 20 to be either a GPIB talker or a GPIB listener or both. The dat and dal enable the talker/listener functions in an inverse order, so the user has to think in "reverse" to understand these functions. To enable the zSBX 20 as a GPIB talker, the listener function must be disabled by writing a one to the dal bit. The GPIB listener function is enabled by disabling the talker function by writing a one to the dat bit. To make the zSBX 20 both a GPIB talker and listener, neither feature can be disabled; rather, write zeros to bits dat and dal, thus enabling both the GPIB talker and listener.

Some users require the zSBX 20 to have dual GPIB addresses for the same board. This useful feature enables the system designer to "partition" the zSBX 20 into two separate devices on the same board. A prime example is when the zSBX 20 is programmed to measure temperature and pressure in an industrial processing plant. One GPIB address pertains to a temperature measurement, the other to a pressure measurement. Writing a one to the enable-dual-primary-addressing (edpa) bit makes the zSBX 20 ignore GPIB address bit A1, giving the board two consecutive GPIB primary addresses. This is not to be confused with GPIB primary-secondary addressing which will be described later. The Upper-Lower-Primary-Address (ULPA) bit in the Address Status Register (discussed in detail later) tells the user which of the dual primary addresses was sent by the controller. The ULPA bit is actually an image of the missing GPIB address bit A1 ignored in the edpa mode.

Note to the user: some system designers will want to build a system that uses a separate GPIB talker and listener at the same GPIB address. The zSBX 20 could be programmed to function in this manner but Ziatech recommends that you avoid this mode of operation for two reasons: (1) non-unique GPIB addresses in the same system are very confusing and (2) the interrupt registers cannot differentiate between a talker or listener being addressed (see the discussion on TMS 9914A interrupt registers later).

The System Reset signal (RESET), generated by the CPU, resets the Address Register such that the zSBX 20 is a single GPIB talker/listener with an address of 0. Therefore, the user's initilization must enable the GPIB talker and listener, and simultaneously write the GPIB address A5-A1. This can be accomplished in the following manner:

(1) Select the GPIB address using the five least significant bits of the on-board DIP switch.

(2) Read the DIP switch via the Address Switch Register at port base+4H.

(3) Mask out the remaining three bits by ANDing the DIP switch value with 1FH (this bit pattern also enables talker/listener and single primary addressing).

(4) Write the masked address to the Address Register at port base+4H.

(5) The zSBX 20 will then be enabled for a single GPIB primary talker/listener address of A5-A1.

(6) You may skip steps 1-5 and write out the correct pattern to the Address Register for your particular system.

Note that the Address Register is not cleared by a hardware or software reset.

The TMS 9914A as a controller talks and listens through use of ton and lon Auxiliary Commands and is not addressed as described above.

ADDRESS SWITCH REGISTER (PORT 34H, R) - GENERAL PURPOSE

The Address Switch Register is actually a read-only port (base+4H) that reads the contents of the on-board DIP switch at pack position 3A. See the schematic for further reference. The contents of the Address Switch Register, that is, the DIP switch setting, is normally written to the Address Register (see the Address Register discussion above) to enable the GPIB talker/listener address. The DIP switch may, however, be used to indicate anything the system designer finds convenient. The TMS 9914A Address Switch Register figure gives the user a useful suggestion on how to implement the DIP switch. This is how the zSBX 20 defines the DIP switch.

D7	D6	D5	D4	D3	D2	D1	D0
SW8	SW7	SW6	SW5	SW4	SW3	SW2	SW1
S.C.	User Defined		GPIB Address				

Figure 8. TMS 9914A Address Switch Register.

As expected, switch positions SW5-SW1 represent the five bit GPIB addresses. The switch positions SW7-SW6 on the zSBX 20 are user defined for one of four possible operations. The switch position SW8, when off, enables the zSBX 20 as the system controller. When SW8 is on, REN and IFC cannot be asserted. The DIP switch closures are inverted twice so that the "on" or "closed" position of a switch represents a binary 1. See the zSBX 20 schematic for details.

The zSBX 20 is shipped from the factory with the DIP switch set up for the zSBX 20 as system controller, shown in Figure 8 above. The GPIB address is four, the user defined DIP switches are set for zero. The Address Switch Register will read: 04H or 00000100B. The drivers are enabled for three-state operation.

ADDRESS STATUS REGISTER (BASE+2H, R) - TALKER/LISTENER

The Address Status Register (ADRST) is a read-only port at base+2H that is only read when the zSBX 20 is used as a GPIB talker/listener. The Address Status Register contains the GPIB address status of the zSBX 20 which is determined by the current GPIB controller-in-charge. The address status is not "latched", that is, it is valid only at the time of reading. This implies that the GPIB controller may change the address status any time during or after reading; therefore, the system designer should carefully study the normal logical protocol of the GPIB, to anticipate any change in address status. Consult the TMS 9914A Address Status Register figure for details.

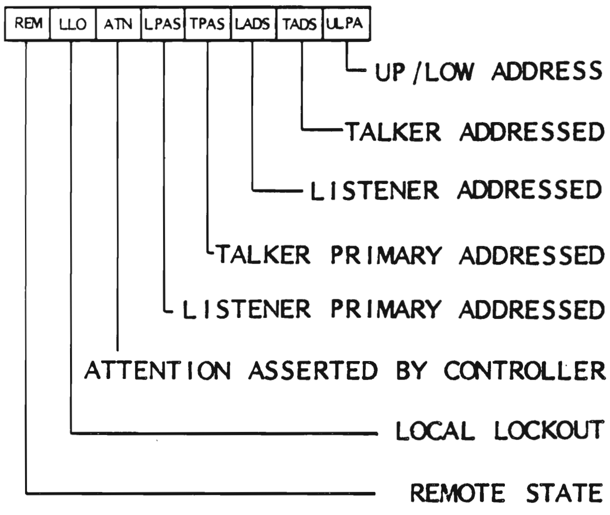


Figure 9. TMS 9914A Address Status Register.

The ULPA bit is used to detect upper or lower dual primary GPIB addresses. Dual primary addressing was discussed in the Address Register section earlier. Please read that subsection before proceeding. The only difference between dual primary addresses is the state of the least significant address bit A1 during addressing. If A1 was low, then the valid address was the lower address and ULPA is cleared. If A1 was high, then the higher address was valid and ULPA is set.

The ULPA bit feature is active regardless if dual primary addressing was selected or not. Once the ULPA bit is set, it can only be cleared by a valid zSBX 20 address with A1 equal to zero, the Interface Clear (IFC) signal on the GPIB, or by removing power. Remember, the ULPA bit only pertains to the last valid zSBX 20 GPIB address sent by the controller. Dual primary addressing should not be confused with secondary addressing which will be discussed later.

The Talker or Listener Primary Addressed State (TPAS or LPAS) bits tell the user that the zSBX 20 GPIB primary talk or listen address has been sent by the controller, and that the GPIB hardware has acknowledged that fact by settling into the talker or listener primary addressed "state". The reader should consult the IEEE 488 Standard for the state diagrams.

The TPAS and LPAS bits are used when a secondary address is required to form a "complete" address. In normal primary addressing mode, a single five bit GPIB address differentiates between thirty-two possible GPIB addresses. Some systems require the user to implement more than thirty-two device addresses; that is, a device within a device or a function within a device needs to be specified. This is typical of multiprocessor systems where many subroutines need to be specified via a second or "secondary" address. Therefore, the TPAS and LPAS tell the user that only the primary address has been received though a secondary address may be required.

The IEEE 488 Standard limits the total number of devices on the bus to sixteen, including the controller. This is a bus loading limitation and not a logical one.

The Talker or Listener Addressed State (TADS or LADS) bits tell the user that the zSBX 20 has been fully addressed by the GPIB controller. If primary addressing only is used, then TADS and TPAS or LADS and LPAS occur at the same time, that is, a single zSBX 20 primary address sent by the GPIB controller completes the addressing state.

If secondary addressing is employed, TADS or LADS tell the user that both primary and secondary address have been received by the zSBX 20. Secondary addressing will be discussed in further detail later.

The TADS or LADS bits do not necessarily mean that the zSBX 20 is ready to talk or listen yet. In order for the zSBX 20 to talk or listen over the GPIB, the Byte-In (BI) and Byte-Out (BO) bits in the Interrupt Status 0 Register must be closely monitored. See the

Interrupt Status 0 Register (INT0) for details.

The Attention (ATN) bit indicates the level of the GPIB Attention line. Attention is only asserted by the GPIB controller currrently in charge. When data is present on the GPIB data bus and Attention is asserted, the data is actually a GPIB bus message such as a talk or listen address. When data is present without Attention, then it is simply data. Advanced GPIB system designers often need to know the level of the Attention line in order to determine the current GPIB state of a device.

The Local Lockout (LLO) bit indicates that the Local Lockout message has been received by the zSBX 20. LLO is a message sent by the GPIB controller to tell the talker/listeners to ignore their front panel controls (if any). This is useful in a system where you need to protect against an accidental switch closure at a control panel or against an inexperienced operator.

The Remote Enable (REM) bit tells the user that the Remote Enable (REN) line on the GPIB has been asserted by the controller and that the zSBX 20 is in the Remote Enable State. The REN line lets a talker/listener know, in this case the zSBX 20, that it is "enabled" to be "remotely" programmed by the controller. Some devices ignore the REN line, that is, they accept control any time from a controller.

The REM bit has another subtle function. Power up time for some systems presents many problems that are not incurred during normal operation. The system controller should power-up, initialize, pulse Interface Clear (IFC), and assert REN. The zSBX 20 can then detect REN via the REM bit. REN tells the zSBX 20 that the controller has powered up successfully and is ready to control the GPIB. The zSBX 20 talker/listener can then safely proceed.

BUS STATUS REGISTER (BASE+3H, R) DEBUGGING

This read only, non-latched register is used to obtain the status of the IEEE 488 bus management lines. The Bus Status Register (BUSTR) is not normally used in a system. Its main purpose would be used in debugging the GPIB should a catastrophic failure occur. All eight lines of the GPIB control lines can be monitored. The bits are positive true logic values of the GPIB management lines. **Please note that this information is obtained from the internal logic of the TMS 9914A and no mechanism is provided to prevent status bits from changing during a read cycle.**

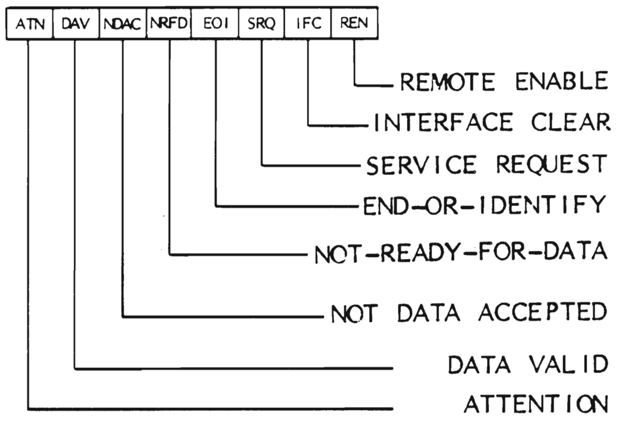


Figure 10. TMS 9914A Bus Status Register.

If the zSBX 20 is configured as the System Controller and is sending IFC, then the IFC bit in this register is not set.

COMMAND PASS-THROUGH REGISTER (BASE+6H, R) - TALKER/LISTENER

This read only port is the Command Pass-Through Register (CPTRG). It monitors the GPIB data lines similarly to the way the Bus Status Register monitors the GPIB control lines. It is a non-latched unqualified image of the GPIB data lines that may be read any time. This register is normally used when the zSBX 20 is a talker/listener, to read secondary addresses, unrecognized commands, and secondary commands.

The register contents are not latched, therefore the user must suppress the handshake, thus forcing the data to be kept stable long enough to read the address or command and then respond correctly. The user must then complete the handshake, allowing new data on the bus.

Handshake manipulation is controlled by the Auxiliary Command Register discussed later.

Handshake suppression is also affected by the Address Pass-Through bit in the Interrupt Mask 0 Register also discussed later.

Although the IEEE 488 Standard does not permit users to define their own commands, provisions for upgrades of the standard are provided by using the Command Pass-Through Register. Thus, the number of possible available commands for future IEEE definition is increased. An interrupt can be generated to prompt the CPU to read the Command Pass-Through Register. See the discussions on the interrupt registers.

When the zSBX 20 is the GPIB Controller, this register is also used to obtain the parallel poll status bits, when conducting a parallel poll. For details, refer to the section on the Parallel Poll Register.

PARALLEL POLL REGISTER (BASE+6H, W) - TALKER/LISTENER

The Parallel Poll Register (PRPR) is only used when the zSBX 20 is a GPIB talker/listener. The parallel poll feature is used when the GPIB controller needs to simultaneously check the request-for-service status of up to eight talker/listeners at the same time. Each of the eight devices will have a dedicated GPIB data line to drive when parallel-polled by the controller. When the zSBX 20 needs to get the attention of the GPIB controller, and the parallel poll feature is being used, the zSBX 20 must save its own user-defined internal status indicating a request for service. When the controller routinely performs a parallel poll, the zSBX 20 must place a "yes" or "no" status bit on its own dedicated GPIB data line. The following text will discuss the mechanism for doing this.

Note to the reader: Most systems use the Serial Poll feature rather than Parallel Poll; Serial Poll is easier to implement, so it is highly recommended that the user implements Serial Poll rather than Parallel Poll.

DIO8	DIO7	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1
PP8	PP7	PP6	PP5	PP4	PP3	PP2	PP1

Figure 11. TMS 9914A Parallel Poll Register.

Whenever the Attention (ATN) and the End-Or-Identify (EOI) line on the GPIB are asserted together by the controller, the contents of the zSBX 20 Parallel Poll Register are asserted on the GPIB data bus. A hardware reset clears the the Parallel Poll Register. A software reset (see the Auxiliary Command Register) must be executed before writing to the Parallel Poll Register.

Obviously, anything can be written to the Parallel Poll Register, but to give each device a dedicated GPIB data line to request service from, only one bit of the parallel poll response byte may be "active" at a time. This means that if the system is using a positive "sense" bit to indicate service requested, the byte written to the Parallel Poll Register must consist of one bit high with the remaining seven bits low. If negative sense is used, then the compliment byte must be written, that is, one bit low and seven bits high. This is the normal PP mode due to the electrical nature of open collector drives with passive pull-ups.

If there are more than eight devices on the GPIB and the system requires parallel polling from each device, devices may share one of the eight GPIB data lines. The system designer must then implement a way to determine which instrument(s) sharing a data line truely requested service. The controller can sequentially interrogate each device or set up another parallel poll subsystem where previously polled devices do not participate in the poll.

The IEEE 488 Standard calls out two subsets of parallel poll capability: an "easy" one and a "not-so-easy" one. These are Parallel Poll Two (PP2) and Parallel Poll One (PP1) respectively.

Parallel Poll Subset PP2

Protocol for PP2 can be simple. With PP2, the GPIB controller can conduct a parallel poll by simply asserting End-Or-Identify (EOI - we are using the "Identify" portion now) while the Attention (ATN) line is asserted; that is, while the controller is actively in charge. Each GPIB device participating in the parallel poll must, within 200 microseconds, send its parallel poll response bit to the GPIB data bus. The controller can then read all the response bits as one data byte and then take appropriate action.

The frequency of polling is determined solely by the GPIB controller. The controller must poll often for busy systems because the talker/listeners do not have any direct means to attract the attention or interrupt the controller, when using parallel poll. The controller must poll frequently to keep close tabs on the system. This is a main consideration when deciding between using parallel versus serial polling.

With serial poll designs, any device may interrupt the controller by asserting the Service Request (SRQ) line on the GPIB. The controller could then "serially" interrogate each device for a serial response byte that describes not only if the device needs service or not, but can also indicate the type of service required with the remaining seven bits. See the Serial Poll Register.

Obviously, "configuring" or getting ready for a parallel poll requires a lot of system considerations. Each device must know what GPIB data line to drive during a parallel poll.

When using the PP2 subset, the initialization routine of the zSBX 20 must write the correct response byte to the Parallel Poll Register, setting the assigned bit if using positive sense or resetting the assigned bit if negative sense is used. All other bits must be complements of the assigned response bit.

Note to the reader: Most systems that use Parallel Poll use PP2.

Parallel Poll Subset PP1

The system designer can build a system where the GPIB controller tells each device how to configure its response byte. The PP1 subset was defined for this purpose.

The four least significant bits of the Parallel Poll Enable (PPE) message are designated S, P1, P2, P3. The Sense (S) bit, corresponding to the fourth GPIB data line, tells the device which polarity the parallel poll response bit must be to be true, that is, an affirmative response. The binary-weighted bits P1, P2, P3, tell the device which GPIB data line to use for the response bit. The remaining four bits not shown, in conjunction with S, P1, P2, P3, make up the PPE message.

When the zSBX 20, as a talker/listener, receives the SPE message, the software must read the message via the Command Pass-Through Register, interpret the S, P1, P2, and P3 information and store the assigned parallel poll response byte in memory somewhere. Up to this point, zero's should have been written in the Parallel Poll Register to avoid confusion.

The suggested protocol for implementing PP1 parallel polling where the zSBX 20 is a talker/listener is as follows:

(1) After power-up and software reset, write 00 to the Parallel Poll Register.

(2) The GPIB controller will address the zSBX 20 to listen.

(3) The controller will send the Parallel Poll Configure (PPC) message. The zSBX 20 will read the command via the Command-Pass-Through Register and then get ready for the PPE message.

(4) The controller will send the "customized" PPE message for the zSBX 20. The zSBX 20 will read the PPE message via the Command Pass-Through Register again, interpret the S, P1, P2 and P3 information, and store the byte for further use.

(5) The zSBX 20 will then be set up for PP1 parallel polling. If the zSBX 20 needs service from the controller, an "affirmative" response byte will be written to the Parallel Poll Register, otherwise the "negative" byte is written.

(6) Whenever the controller requests a parallel poll response byte, the ATN and EOI lines are asserted by the controller. The "yes" or "no" response in the zSBX 20 Parallel Poll Register is automatically placed on the GPIB data bus.

(7) If or when the controller re-addresses the zSBX 20 to listen and sends the Parallel Poll Disable (PPD) message, the zSBX 20 must not write an affirmative response byte into the Parallel Poll Register until the zSBX 20 is re-enabled by the controller by repeat-

ing steps (3) through (4). This feature allows several devices to share a parallel poll response line by disabling the devices that are known not to need service and enabling the devices in question.

(8) If or when the controller sends the Parallel Poll Unconfigure (PPU) message, the zSBX 20 may interpret this message to imply that no more parallel poll activity will be taking place until the re-configure (PPC) message gets sent again by the controller.

Parallel Poll versus Serial Poll

A Parallel Poll service request differs from the Serial Poll service request in these ways:

(1) A device using the parallel poll facility should be assigned its own dedicated bus line to send its request, whereas devices using the serial poll facility (i.e., SRQ) must be addressed individually to send an identifying service request byte. Parallel Poll saves the talk addressing time and can identify up to eight devices at once.

(2) Devices using the serial poll facility (SRQ) can request service from the controller any time a device requires service, whereas service requests sent via the parallel poll facility can only be sent when solicited by the current controller. Thus, if speed in servicing requests is of great importance and there is little GPIB bus activity between requests (permitting frequent parallel polls by the controller) servicing of requests should be done by the parallel poll methods. **By far, however, the serial poll method is the easiest to use and is applicable for the majority of GPIB systems.**

(3) The serial poll mechanism implicitly tells the device that its request has been seen by the controller and it may stop asserting SRQ. Parallel Poll has no equivalent mechanism and the system software in both the device and the controller must explicitly set up some convention to inform the device that its parallel poll response has been recognized.

Keep in mind that protocol for the IEEE 488 Bus has not been defined, that is, it is left up to the designer. Bus messages, and the effect thereof, on GPIB devices, have been defined such that almost all GPIB devices are compatible when a reasonable systematic protocol is designed.

Parallel Poll GPIB Drivers

Both parallel poll subsets require that open collector GPIB transceivers be used to return the status byte when polled. The GPIB driver, pack 1B, is enabled for open collector operation automatically during a parallel poll. This is accomplished by the 75453 at pack location 1E. During normal operation, the drivers operate in three-state mode for the fastest data transfers. See the discussion on 'GPIB Transceivers' in this section.

SERIAL POLL REGISTER (BASE+5H, W) TALKER/LISTENER

The serial poll facility of the GPIB is the easiest and most useful polling method used on the GPIB. The main distinction of serial polling over parallel polling is that in serial polling, each talker/listener can interrupt the controller at any time via the Service Request (SRQ) line. When parallel polling has been implemented the controller must periodically poll or interrogate the bus to check device status.

DIO8	DIO7	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1
S8	rsv1	S6	S5	S4	S3	S2	S1

Figure 12. TMS 9914A Serial Poll Register.

When an SRQ is generated by a device using serial poll, the controller sends the Serial Poll Enable (SPE) message to the GPIB. Each device participating in the serial poll, regardless of whether it generated a service request or not, goes into the Serial Poll Mode State (SPMS) where each device must get ready to participate in the serial poll.

The controller then sequentially or "serially" runs down a device address list, addresses a device to talk, and then listens to or reads the device response called a "Serial Poll Response Byte". If the polled device truely generated a service request (realizing that more than one device could have requested service), the device must assert, as a minimum, bit 7 of its serial poll response byte. If bit 7 is not asserted then the controller knows the device did not request service. The controller keeps polling until all the devices in the device list have been polled.

The remaining seven bits of the serial poll response byte may contain user-defined information such as the type of service requested or some other machine status. This makes the serial poll mechanism the most popular of the GPIB polling techniques.

The Serial Poll Register (SPOLR) is used only when the zSBX 20 is a talker/listener. The zSBX 20 must store its serial poll response byte in this register. When the GPIB controller sends a SPE message to the bus followed by the zSBX 20 talk address, the contents of the Serial Poll Register are placed onto the GPIB data bus. The zSBX 20 will keep asserting the response byte until the controller re-addresses another device to talk or sends the Serial Poll Disable (SPD) message. The controller must read the serial poll response byte only once and then continue the serial poll.

The zSBX 20 can generate a service request by two methods. The easiest and most used method is via the auxiliary command called "Request Service Two" (RSV2) that is written to the Auxiliary Command Register (see the discussion on Auxiliary Commands). This method should be used whenever possible. When the service request has been generated, the controller will eventually perform a serial poll. The suggested protocol follows:

(1) The zSBX 20 will request service (asserts SRQ) via the RSV2 command.

(2) The GPIB controller will send a Serial Poll Enable (SPE) message.

(3) The controller will address the zSBX 20 to talk.

(4) The controller will deassert the Attention (ATN) line, and the zSBX 20 serial response byte will be automatically placed on the GPIB data bus. The SRQ line will be automatically cleared after being read.

(5) The controller will read the response byte and the zSBX 20 will generate a Serial Poll Active State (SPAS) interrupt, to the on-board 8085 CPU, if enabled (see the Interrupt Status and Mask Register).

(6) The controller will reassert ATN and take control of the GPIB again. A second SPAS interrupt will be generated on the zSBX 20 if enabled.

(7) The controller will continue polling the remaining devices on the GPIB. The serial poll is terminated by the Serial Poll Disable (SPD) message sent by the controller after the controller polls the last device.

The second method of requesting service is accomplished by writing a 1 to the RSV1 bit in Serial Poll Register. The same protocol is used as with RSV2 except that in order for the zSBX 20 to ever generate another service request, the RSV1 bit must first be cleared by writing a 0 to it. The RSV1 bit would then be ready to be set again.

When the zSBX 20 has not requested service, but is serial polled by the controller as a result of another device requesting service, the response byte is transferred to the GPIB data bus as in the two cases above. The SPAS bit in the Interrupt Status Register 0 (INT0) is never set, and thus never generates an interrupt. Also, bit 7 of the Serial Poll Response byte is not asserted.

DATA IN REGISTER (PORT 7H, R) CONTROLLER, TALKER/LISTENER

The zSBX 20 reads all data from the GPIB via the Data In Register (DIN). The GPIB hardware on the zSBX 20 was designed such that it will not let you lose a single byte of data before the CPU can read the Data In Register. The GPIB hardware will suppress the three wire handshake either automatically or under software control, allowing an infinite length of time for the CPU to read the in-coming data.

The Data In Register will only accept a byte of data from the GPIB if the previous byte was read, and only if any previous data "hold-offs" (see the Auxiliary Commands) have been removed by the processor. Also, data can only be read from the GPIB if the zSBX 20 has been addressed to listen, as in the case when the zSBX 20 is a talker/listener or if the zSBX 20 put itself in a "talk-only" mode as in the case where it is the GPIB controller.

The following suggested protocol can be used when the zSBX 20 is a GPIB listener:

(1) When the GPIB controller addresses the zSBX 20 to listen, the My Address (MA) and My Address Change (MAC) interrupts will occur if enabled (see the Interrupt Register discussions). The zSBX 20 will be put in the Listener Primary Addressed State (LPAS) and Listener Addressed State (LADS).

(2) The controller will remove control by deasserting Attention (ATN)

(3) The active talker, which could be the controller or any other talking device, will send a valid data byte. The Byte-In (BI) interrupt will be generated if enabled. The CPU must then read the byte from the Data In Register.

(4) Step (3) will be repeated for each data byte sent by the active talker.

(5) After the last data byte is sent by the talker and subsequently read from the Data In Register, the controller will un-address the zSBX 20 from listening by a Universal Unlisten (UNL) message. A MAC interrupt will be generated if enabled.

Making the zSBX 20 a listening controller is a little more difficult. The zSBX 20 must first be initialized as a controller. The following protocol is suggested:

(1) Generate a chip reset and clear reset via the Software Reset (swrst) auxiliary command.

(2) Force the zSBX 20 to take control of the GPIB and to send Interface Clear (IFC) to the GPIB by issuing the Send Interface Clear (SIC) auxiliary command. The zSBX 20 will then be the system controller.

(3) Put the zSBX 20 into the "talk only" mode by issuing the talk only (ton) auxiliary command. This will complete the zSBX 20 controller initialization. The talk only mode can be considered the default controller mode.

(4) Put the zSBX 20 into the "listen only" mode by issuing the listen only (lon) auxiliary command. A GPIB controller should always default to the talk only mode so it can talk or send GPIB messages.

(5) Force the zSBX 20 to release control and deassert ATN via the Go To Standby (gts) auxiliary command.

(6) The zSBX 20 may then listen to the GPIB data bus. The CPU can read the Data In Register as in step (3) and (4) above with each byte predicated by a BI interrupt if enabled.

(7) After the last byte is read by the zSBX 20, the zSBX 20 can re-take control of the GPIB by issuing the Take Control Synchronously (tcs) auxiliary command. The ATN line will be reasserted.

(8) The zSBX 20 should be put back into the talk only mode.

Note that the zSBX 20 has a separate Data In Register and a Data Out Register which means that GPIB data can be read and written without destroying the contents of the opposite register.

Several data "hold-off" modes can be selected via the auxiliary commands. These will be discussed in detail later. The main function of data hold-off is to suppress the handshake on the GPIB long enough to let the CPU examine the data byte being listened to.

The data may be just data, but more often it is an unrecognized command such as a Parallel Poll Enable or a secondary address. Messages or commands are different than data, for the controller is asserting ATN and the GPIB hardware will normally accept the message without waiting for the CPU to read the Data In Register. A "held-off" data byte or message is "unheld" or "released" by one of the release hold-off auxiliary commands also discussed later.

DIO8	DIO7	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1

Figure 13. TMS 9914 Data In Register.

DATA OUT REGISTER (BASE+7H, W) CONTROLLER, TALKER/LISTENER

The Data Out Register (DOUT) is used by the zSBX 20 to send or output data to the GPIB data bus. When Attention (ATN) is asserted on the GPIB, the data becomes a command or message. Only the controller currently in charge of the bus can send commands. Every data transaction via the Data Out Register is accompanied by a handshake.

The Byte Out (BO) interrupt in the Interrupt Status Register is used to tell the CPU that the previous byte sent by the zSBX 20 was accepted by all other devices on the GPIB, that is, the handshake was completed. Also, when the zSBX 20 is first addressed to talk by the current active controller, and ATN is unasserted, the BO bit goes high and a BO interrupt is generated if enabled. This action tells the CPU that it is okay to write out to the Data Out Register. The BO bit will not be set again until the current byte was accepted by all GPIB devices.

The system designer must make provisions for terminating data transfers, because the last byte written to the Data Out Register that is accepted by the GPIB will set the BO bit. To prevent the CPU from blindly trying to write another byte to the Data Out Register, some convention must be invented to terminate data strings. The current talker must know either the number of bytes it must send, and/or the last character (end-of-string character) sent. The active listener(s) should also know this. The End-Or-Identify line on the GPIB will also serve the purpose of indicating the "End" of a data transfer. The EOI line should be asserted by the talker with the last byte to tell all listeners to not expect any more data. See the Auxiliary Commands for asserting EOI.

When the zSBX 20 is used as a GPIB controller, the following protocol should be observed to initialize the Data Out Register for sending data.

(1) Execute a software reset and clear reset via the swrst auxiliary command.

(2) Send Interface Clear (IFC) and take control (assert ATN) via the Send Interface Clear (sic) auxiliary command. Do not forget to clear the command.

(3) The Byte Out (BO) bit will then be set, indicating a "receptive" GPIB data bus. Reading the BO bit in the Interrupt Status Register 0 , it will clear the BO bit, but the Data Out Register will still be ready.

(4) Put the zSBX 20 in the talk only mode via the talk only (ton) auxiliary command.

(5) A byte may be written to the Data Out Register providing that the BO bit was set as a result of the sic auxiliary command and that nothing else was written to the Data Out Register prior to that point. A software polling loop should be implemented to wait for BO to be set, keeping in mind that once the BO bit is read, it is cleared by the read operation.

(6) When BO is set, and a byte is written to the Data Out Register, the byte gets sent to the GPIB as a command and not data. This is because ATN was asserted by the sic auxiliary command. Be sure the byte is indeed a valid GPIB message that can be interpreted by the devices on the GPIB.

(7) To send data to a device on the GPIB, the zSBX 20 must first address the correct device(s) to listen to the data. After the devices have been addressed to listen, the zSBX 20 must remove the ATN line by issuing the Go To Standby (gts) auxiliary command.

(8) When BO is set, a data byte may be written to the bus. All devices will complete the handshake, causing BO to be set again, but only the active listeners will actually "read" the data.

(9) The controller should take control again by asserting ATN via the Take Control Asynchronously (tca) auxiliary command.

INTERRUPT MASK/STATUS REGISTERS (BASE+0,1H, R/W) CONTROLLER, TALKER/LISTENER

The Interrupt Mask and Status Register are the most used registers by the zSBX 20 when interfacing to the GPIB, whether or not interrupts are even used. **These interrupt registers should be studied in great length in order to understand the operation of the GPIB interface.** The Interrupt Registers are usually the first and last registers read when using the GPIB interface, and usually point to the next operation, if any, to perform.

The interrupt registers operate independently of the mask register. No interrupt is generated if the corresponding mask bit is set to a zero (i.e., masked off). The status registers are double buffered such that any event which causes a status register to change during a CPU read cycle will be stored and will set the corresponding bit at the end of the read cycle. The previously set bits are cleared at the end of the read. The interrupt status registers are also cleared by either a hardware reset or a Software Reset (swrst) auxiliary command.

With the exception of INT0 and INT1, each bit is set when the corresponding event occurs. Once set, the corresponding register must first be read, and then the interrupt condition be false and true again before that status bit will be set again. However, INT0 and INT1 are set only on the condition that at least one event occurs in status registers 0 or 1, and that the corresponding bit in the interrupt mask register is also set, that is, masked on so that interrupts are enabled.

Note that the INT0 and INT1 bits are only cleared when the interrupt register causing the interrupt is read. Also note that an interrupt is enabled, that is, masked on when the mask bit is set to a one. Both mask registers are cleared by a hardware reset, but not by Software Reset.

X	X	BI	BO	END	SPAS	RLC	MAC	M
INT0	INT1	BI	BO	END	SPAS	RLC	MAC	S

Figure 14. TMS 9914A Interrupt Mask/Status Register 0.

The Interrupt 0 (INT0) and Interrupt 1 (INT1) bits indicate that a condition in the Interrupt Status Register 0 or Interrupt Status Register 1 caused an interrupt, respectively. Obviously, at least one of the conditions must have been enabled to generate an interrupt by having set a corresponding mask bit at an earlier time. Any GPIB interrupt, when enabled, will generate a RST6.5 interrupt to the on-board CPU. The interrupt service routine should first disable all 8085 interrupts via the Disable Interrupt (DI) instruction, to prevent further interrupts from possibly being serviced, and then read the INT0 and INT1 bits to see what GPIB activity caused the interrupt.

The Byte In (BI) bit is set when a data byte or a command has been received by the GPIB Data In Register. The primary function of the BI bit is to tell the on-board CPU to hurry up and read the Data In Register so that another byte may be input. The BI bit is reset when the CPU reads the INT0 register. The BI bit will not be set when the zSBX 20 is in the shadow handshake mode. See the auxiliary commands discussion.

The Byte Out (BO) bit is set when the Data Out Register is ready to be loaded with a data byte or GPIB command. It basically tells the on-board CPU that all the GPIB devices have accepted the last byte and/or that each device is ready for another byte or command. This bit is also reset by reading the INT0 Register.

The End (END) bit indicates that the byte just received in the Data In Register was the last byte, indicated by the End-Or-Identify (EOI) line on the GPIB being asserted by the active talker. The talker could have been the controller.

The Serial Poll Active State (SPAS) bit gets set twice during serial polling. It is read when the zSBX 20 is a talker/listener only. If the corresponding mask bit is set, it also will generate two interrupts with each set condition. The SPAS bit first gets set when the controller reads the serial poll response byte from the zSBX 20.

When the controller reasserts the Attention line after reading the zSBX 20 response byte (usually to poll another device or disable serial poll), the second setting of SPAS occurs. If the zSBX 20 never requested service, but gets serial polled as a result of another device requesting service, the SPAS bits will not get set, and no corresponding interrupt will be generated. Keep in mind that the Serial Poll Register contents will still be read by the controller.

The Remote-To-Local Change (RLC) bit gets set whenever the controller in charge sends a Remote or Local message (or REN) to the zSBX 20. The RLC is only used when the zSBX 20 is a talker/listener. The RLC message implies that an instrument may respond to its "front panel" controls if the front panel was previously disabled by the GPIB controller. RLC does not mean anything inherent to the zSBX 20 because it is a microcomputer and the microcomputer always has access to the GPIB hardware as long as it is running. RLC would only have relevance to the zSBX 20 if it was being used to interface to a human interface such as a keyboard or control panel. The zSBX 20 could then interpret the RLC message and subsequent RLC bit setting to "return to local" control of the operator by scanning and responding to the control panel again. The power up configuration for a GPIB instrument is normally a local control state that may be removed or superseded by the GPIB controller.

The My Address Change (MAC) bit is only read when the zSBX 20 is a talker/listener. The MAC bit is set whenever the zSBX 20 address status has been changed by the GPIB controller. The MAC should be the first bit examined when any change in talker/listener addressing is suspected by the zSBX 20. As an example, when the controller addresses the zSBX 20 to listen, the MAC bit is set. The MAC bit is reset by reading the INTO Register. The zSBX 20 should then go into some listen routine designed by the user. When the controller unaddresses the zSBX 20 to listen, that is, the zSBX 20 is no longer an active listener, then MAC is set again.

GET	ERR	UNC	APT	DCAS	MA	SRQ	IFC	M
GET	ERR	UNC	APT	DCAS	MA	SRQ	IFC	S

Figure 15. TMS 9914A Interrupt Mask/Status Register 1.

The Group Execute Trigger (GET) bit is used only when the zSBX 20 is a talker/listener to indicate when the GPIB controller sent the Group Execute Trigger message to the GPIB. The zSBX 20 must have previously been addressed to listen by the controller. The GET message can be used as a GPIB system synchronization signal where multiple listeners can respond to a command at the same instant. This can be useful in a system where the controller needs to start or stop a group of real-time clocks on the GPIB.

The Error (ERR) bit is used to detect errors in the handshake sequence. When the zSBX 20 is going to send a byte to the GPIB, and the Not-Ready-For-Data (NRFD) and Not-Data-Accepted (NDAC) lines are both sense high indicating an invalid source handshake, the ERR bit is set and the byte in the Data Out Register is not sent. This is not a typical condition in GPIB systems, and thus the ERR bit usually indicates that no devices in the system are addressed to listen.

The Unrecognized Command Group (UNC) bit is used to tell the zSBX 20 that a GPIB command sent by the controller is not known by the TMS 9914A hardware. This means that the interpretation of the command must be handled by software. The UNC bit is only used when the zSBX 20 is a talker/listener. The zSBX 20 could be an inactive controller currently acting as a talker/listener. The following three bus messages that set the UNC bit are:

(1) Take Control (TCT) if the zSBX 20 is addressed to talk.

(2) My Secondary Address (MSA) if the Pass Through Next Secondary (pts) auxiliary command was issued previously.

(3) Unrecognized Universal Command Groups (UUCG) or Unrecognized Addressed Command Group (UACG). See the IEEE 488 Standard Section 2.13.

The Address Pass-Through (APT) bit is used by the zSBX 20 only when it is a talker/listener. The APT bit tells the zSBX 20 that an extended or secondary address was sent by the GPIB controller. To enable the zSBX 20 for secondary addressing, the APT bit in the Interrupt Mask Register 1 must be set.

When any secondary address gets sent by the controller, an APT interrupt is generated and an automatic Accepted Data State (ACDS) holdoff is initialized. No more GPIB bus activity will take place until the zSBX 20 CPU reads the secondary address from the Command Pass Through Register and issues one of two auxiliary commands.

If the CPU recognizes the secondary address as a valid secondary address, then the data holdoff is released by sending the Data Accepted Release (dacr) auxiliary command with the most significant bit set high. This action completes the handshake, allowing GPIB activity to continue, forcing the zSBX 20 to enter the completed address state.

If the CPU does not recognize the secondary address as being valid then a dacr auxiliary command is issued with the most significant bit set low. This forces the zSBX 20 to complete the handshake but not enter the completed addressed state.

Secondary addressing is also useful for sending information other than addresses as the creative system designer might imagine.

The Device Clear Active State (DCAS) bit is used only when the zSBX 20 is a talker/listener. The DCAS tells the zSBX 20 that the GPIB controller sent the Device Clear (DCL) message. DCL is sent by the controller to "clear" all or a subset of talker/listener on the bus individually selected by prior listen addressing.

The effect of DCL on a device is a function of system design. It is not meant to be a "reset" but could indirectly be used in that manner. System designers may implement the DCL message to force the listening device to enter the power on (pon) state thus forcing all states into an idle condition.

The system designer could also define the DCL function to force listening devices into any "non-obtrusive" state. As an example, if the zSBX 20 were used as a GPIB data logging system, DCL might be implemented to reset any internal software counters or timers, but not to clear data.

The My Address (MA) bit is only used when the zSBX 20 is a talker/listener. The MA bit tells the zSBX 20 that it has been addressed to talk or listen by the controller. The MA bit is not set after the Serial Poll Enable (SPE) message has been sent by the controller, that is, during a serial poll sequence. The My Address Change (MAC) bit, however, is effected. See the MAC bit discussion above.

The Service Request (SRQ) bit is used by the zSBX 20 when it is a controller only. The SRQ bit is set whenever a device on the GPIB requests service from the zSBX 20 controller by asserting the SRQ line.

The Interface Clear (IFC) bit is only used when the zSBX 20 is a talker/listener. The IFC bit tells the zSBX 20 when the system controller asserts the IFC line on the GPIB. IFC is normally pulsed only during power up time and/or reset. When the zSBX 20 detects an IFC pulse, the software should completely reinitialize the system. All GPIB functions should be put in their idle state.

Note to the reader: When the GET, UNC, APT, DCAS, and MA bits have been enabled to generate an interrupt to the CPU, and one of these states occurs, thus generating an interrupt, an Accept Data State (ACDS) holdoff is automatically initiated. All GPIB activity is then temporarily suspended and the handshake is "suppressed". The on-board CPU must interpret the cause of the interrupt, take appropriate action depending on the system, then complete the handshake by issuing the release data holdoff (dacr) auxiliary command. This necessary feature of the zSBX 20 gives the on-board processor time to respond to interrupts without losing GPIB information.

AUXILIARY COMMAND REGISTER (BASE+3H, W) CONTROLLER, TALKER/LISTENER

The Auxiliary Command Register (AUXCMD) provides many of the special features of the zSBX 20 GPIB interface. An auxiliary command is issued by writing the command byte to the Auxiliary Command Register. Each auxiliary command is explained in detail in the following subsection. Please see the TMS 9914A AUXILIARY COMMANDS figure.

C/S	XX	XX	f4	f3	f2	f1	f0

Figure 16. TMS 9914A Auxiliary Command Register.

TMS 9914A AUXILIARY COMMANDS

A number of the auxiliary commands are the CLEAR/SET type. If a command is loaded with the C/S bit equal to one, the function is selected and remains selected until the code is loaded with the C/S bit equal to zero. The Talk Only (ton) and Listen Only (lon) commands operate in this manner.

Other commands, such as the Force EOI (feoi) and Release RFD Holdoff (rhdf) commands, have a pulsed mode of operation where the C/S bit is not applicable (na), as shown in the auxiliary command table.

The Force Group Execute Trigger (fget) and Return To Local (rtl) commands can operate in either CLEAR/SET or pulsed modes. If the "fget" command is loaded with C/S equal to zero, a pulse appears at the trigger output of the TMS 9914A. If the command is loaded with the C/S bit equal to one, the trigger output goes high until the command is issued again with C/S equal to zero.

If the Return To Local (rtl) command is issued with C/S equal to zero, the REM status bit in the Address Status Register is reset. REM can be set again at any time by a REN command from the GPIB controller in charge.

If the rtl command is issued with C/S equal to one, the REN bit is cleared and cannot be set until the rtl command is issued again with C/S equal to zero. The rtl command has no effect if the Local Lockout (LLO) mode has been selected by the GPIB controller.

C/S	F4	F3	F2	F1	F0	MNENONIC	FUNCTION
0/1	0	0	0	0	0	swrst	Software Reset
0/1	0	0	0	0	1	dacr	Release ACDS Holdoff
na	0	0	0	1	0	rhdf	Release RFD Holdoff
0/1	0	0	0	1	1	hdfa	Holdoff On All Data
0/1	0	0	1	0	0	hdfe	Holdoff On EOI Only
na	0	0	1	0	1	nbaf	New Byte Avail. False
0/1	0	0	1	1	0	fget	Force Group Exec. Trig.
0/1	0	0	1	1	1	rtl	Return To Local
na	0	1	0	0	0	feoi	Send EOI w/Next Byte
0/1	0	1	0	0	1	lon	Listen Only
0/1	0	1	0	1	0	ton	Talk Only
na	0	1	0	1	1	gts	Go To Standby
na	0	1	1	0	0	tca	Take Control Asynch.
na	0	1	1	0	1	tcs	Take Control Synch.
0/1	0	1	1	1	0	rpp	Request Parallel Poll
0/1	0	1	1	1	1	sic	Send Interface Clear
0/1	1	0	0	0	0	sre	Send Remote Enable
na	1	0	0	0	1	rqc	Request Control
na	1	0	0	1	0	rlc	Release Control
0/1	1	0	0	1	1	dai	Disable All Interrupts
na	1	0	1	0	0	pts	Pass Thru Next Sec.
0/1	1	0	1	0	1	stdl	Set T1 Delay
0/1	1	0	1	1	0	shdw	Shadow Handshake
0/1	1	0	1	1	1	vstdl	Very Fast T1 Delay
0/1	1	1	0	0	0	rsv2	Request Service 2

Figure 17. TMS 9914A Auxiliary Commands.

Software Reset (swrst) 0/1XX00000

The Software Reset Command is issued when the zSBX 20 is a talker/listener or a controller. In both cases, the swrst must be issued at power up and/or system reset to begin initialization of the GPIB hardware. When the zSBX 20 is a talker/listener, the swrst should be executed when the GPIB controller asserts the Interface Clear (IFC) line.

Issuing the swrst command with C/S equal to one causes all input to the GPIB hardware to be ignored. The Serial Poll Register, Parallel Poll Register 0 and 1 are cleared. Also, when the swrst is set, the GPIB hardware is forced into the following states:

SIDS	Source Idle State
CIDS	Controller Idle State
AIDS	Acceptor Idle State
LOCS	Local State
TIDS	Talker Idle State
NPRS	Negative Poll Response State
TPIS	Talker Primary Idle State
PPIS	Parallel Poll Idle State
LIDS	Listener Idle State
SPIS	Serial Poll Idle State
LPIS	Listener Primary Idle State

When a power-on or push-button reset is generated on the zSBX 20, the swrst is automatically generated internal to the GPIB hardware, forcing the swrst command into its set condition. Whenever the swrst command is set, either by software or automatically, it must be cleared by re-issuing the swrst command with C/S equal to zero.

Release ACDS holdoff (dacr) 0/1XX00001

The Release Accepted Data State holdoff command (dacr) is used when the zSBX 20 is a talker/listener. The dacr command is issued to complete a handshake that was put into Data Accepted Holdoff as the result of receiving an unrecognized command, secondary address, device trigger or device clear message. When the zSBX 20 is receiving data from the GPIB, and an ACDS holdoff occurs, the talking device holds the data in its valid state giving the on-board CPU unlimited time to read and process the valid data. When the CPU has decided what to do with the data, it completes the handshake via the dacr command.

The dacr command is used in two modes: secondary addressing, and primary addressing only. When Address Pass Through (APT) interrupt is enabled, an ACDS holdoff will occur whenever a secondary address is received. If the CPU recognizes the secondary address as being valid, the CPU must issue the dacr command with C/S equal to one. If the secondary address is invalid, the dacr must be issued with C/S equal to zero. In any case, the handshake is complete; however, the zSBX 20 remains unaddressed for invalid secondary addresses.

When secondary addressing is not being used, ACDS holdoffs due to unrecognized commands are released by issuing the dacr command with C/S equal to zero.

Release RFD Holdoff (rhdf) naxx00010

The Release Ready for Data Holdoff command is used by the zSBX 20 when it is both a talker/listener and controller. The rhdf command is issued to release any data holdoff caused by a hdfa or hdfe auxiliary command. The C/S bit is not applicable.

There is an important distinction between a RFD and ACDS holdoff. The ACDS holdoff is used to give the on-board CPU time to read a GPIB command. The byte remains valid as long as the CPU needs to process the data and issue the dacr command completing the handshake. Bus activity is terminated.

A RFD holdoff occurs after a byte was accepted and read by the CPU. No other data is sent. No more valid data is allowed on the bus until the rhdf command is issued, completing the handshake. In both cases, GPIB activity is suppressed, but in the ACDS case, current data is being evaluated, and in the RFD case there is no current valid data.

Holdoff On All Data (hdfa) 0/1XX00011

The hdfa command is used by the zSBX 20 when it is both a talker/listener and a controller. When the hdfa command is issued with C/S equal to one, a Ready For Data Holdoff (RFD) is generated with every GPIB data byte. The handshake must be completed by issuing the rhdf command. GPIB commands, that is when Attention is asserted, are not affected by this command. The hdfa command is unasserted by issuing the command with C/S equal to 0.

Holdoff On EOI Only (hdfe) 0/1XX00100

The hdfe command is used by the zSBX 20 when it is a talker/listener or a controller. When the hdfe command is issued with C/S equal to one, a Ready For Data Holdoff (RFD) is generated for the last GPIB data byte indicated by the End-Or-Identify (EOI) signal. This holdoff gives the CPU time to respond to a string of data terminated by EOI before allowing another string to be received. The holdoff must be released by issuing the rhdf command, thus completing the handshake. The hdfe mode is unasserted by issuing the hdfe command with (C/S=0).

Set New Byte Available False
(nbaf) naXX00101

The nbaf command is used by the zSBX 20 only in the talker modes. The C/S bit is not applicable. The main function of nbaf is to "back out" of a GPIB data send sequence by cancelling the data previously written to the Data Out Register. The protocol is as follows:

(1) Assume the zSBX 20 has been an active talker, sending a string of data bytes to the addressed listeners.

(2) Assume that in midstream of this string of data that a particular byte sent out by the zSBX 20 is accepted by all devices, that is, the handshake is completed. If for some system-dependent reason, the last byte sent demanded immediate attention of the GPIB controller, such as an error condition, the controller would take immediate control of the GPIB (assert Attention).

(3) The zSBX 20 would still see the Byte Out (BO) bit set in the Interrupt Status Register 0, indicating the need to write out the next byte in the data string to the Data Out Register. If this happens, the Data Out Register cannot be written again until that last byte written, that is, the byte following the error, is accepted by the GPIB listeners. Assume that happened:

(4) When the controller releases Attention again, the last byte written to the Data Out Register will be sent automatically to the GPIB, requiring all listeners to accept the byte. If this is not desirable, that is, if as a result of the previous byte sent, a different byte than the one currently in the Data Out Register is needed, the zSBX 20 may "change its mind" by issuing a nbaf command.

(5) When the nbaf command is issued, the current byte is not "removed" or reset, but the "validity" of the byte is cancelled, that is, when ATN is released, Data Valid (DAV) will not be asserted, until a new byte is written to the Data Out Register. After nbaf is issued, the Byte Out is also set again, indicating that the Data Out Register is free to be written again.

A well organized system designer should be able to configure a system such that the nbaf command is not necessary. This command is only used in a deadlock situation. Also, a GPIB controller should not take control in the middle of a block transfer. Provision is normally provided to terminate a block transfer, such as an End-Of-String (EOS) character, byte counter, or End-Or-Identify (EOI).

Force Group Execute Trigger
(fget) 0/1XX00110

This is a general purpose command. The state of the Trigger output from the TMS 9914A is affected when this command is issued. If the C/S bit is zero, the line is pulsed high for approximately 5 clock cycles (1 microsecond at 5 MHz). If C/S is one, the TRIGGER line goes high until "fget" is sent with C/S equal to zero. No interrupts or handshakes are initiated and only the TMS 9914A is affected. The Trigger Output pin is not used on the zSBX 20.

Return To Local (rtl) 0/1XX00111

This is also a general purpose command. When the rtl command is issued, provided the local lockout (LLO) has not been previously enabled, the remote/local status bit is reset and an interrupt is generated (if enabled) to inform the on-board CPU that it should respond to front panel controls if applicable. If the C/S bit is set to one, the rtl command must be cleared by issuing the rtl command with C/S equal to zero, before the device is able to return to remote control. If C/S is set to zero, the device may return to remote without first clearing rtl.

Force End-Or-Identify (feoi) naXX01000

This is used by the zSBX 20 when it is a talker or talking controller. The command causes the End-Or-Identify (EOI) message to be sent with the next data byte. The EOI line is then reset.

Listen Only (lon) 0/1XX01001

The Listen Only command is used by the zSBX 20 to set up itself as a listener. This should only occur if the zSBX 20 is placed in a system where there is no controller or when the zSBX 20 is the controller.

After the lon command is issued with C/S equal to 1, the zSBX 20 becomes a self-addressed listener indicated by the Listen Addressed State (LADS) bit being set in the Address Status Register. The listener feature must not be disabled, that is, the Disable Listener (dal) bit in the Address Register must not be set.

The on-board CPU may read data from the GPIB via the Data In Register (listen activity) whenever the Byte In (BI) is set in the Interrupt Status Register 0. The lon command is reversed by issuing the command with C/S equal to zero, or issuing ton Auxiliary Command.

Talk Only (ton) 0/1XX01010

The Talk Only (ton) command is analogous to the lon command. It is used by the zSBX 20 to address itself to talk although no real addressing takes place. Use the ton command only when the zSBX 20 is the controller or if there is no controller in the system. An example of a no-controller system would be a reporting (talking) voltmeter and a printer set up as a listener.

To enable the talk only mode, issue the ton command with C/S equal to one. The ton should be removed by issuing the command with C/S equal to zero. After the ton mode has been enabled, the Byte Out (BO) bit in the Interrupt Status Register 0 is asserted to let the on-board CPU know that it is okay to write to the Data Out Register. When all the listening devices have completed the handshake, the BO bit is set again. The Disable Talker (dat) bit in the Address register must not be set, that is, do not disable the talk ability of the GPIB hardware.

Note to users: The ton and lon commands were designed to be used with systems without a controller. However, when the zSBX 20 is the GPIB controller, the ton and lon functions are used to set itself to talk and listen respectively. The user should be aware that if the zSBX 20 as a controller is sending GPIB messages such as Untalk (UNT), Unlisten (UNL), or Other Talk Address (OTA), the zSBX 20 talk or listen status is subject to those messages. For example, UNT will reset the ton feature, taking the zSBX 20 out of talk activity. Note also that the ton and lon auxiliary commands are mutually exclusive by the GPIB hardware, that is, the most recently issued command will be the one in effect.

Go To Standby (gts) naXX01011

This command instructs the zSBX 20 to unassert the Attention (ATN) line, thus going to standby. This is a GPIB controller function only.

Take Control Asynchronously
(tca) naXX01100

This command instructs the zSBX 20 to reassert ATN as controller in charge. The command is executed immediately and data corruption or loss may occur if a talker/listener is in the process of transferring a data byte. If a controller has been talking, then use 'tca' after the last BO interrupt to reassert the ATN line without corrupting data. A BO interrupt is generated when the zSBX 20 has entered the controller active state.

Take Control Synchronously
(tcs) naXX01101

This command is used by the controller in charge to set the Attention line true and to gain control of the GPIB. If the controller is not a true listener, the shadow handshake command must be used to monitor the handshake lines. This forces the zSBX 20 to synchronize with the talker/listeners, sending

ATN true only at the end of a byte transfer. This ensures that no data will be lost or corrupted. A BO interrupt is generated when the TMS 9914A has entered the controller active state.

Request Parallel Poll (rpp) 0/1XX01110

This command is used by the controller in charge to send the parallel poll command over the GPIB (the zSBX 20 must be in the Controller Active State so that the Attention line is asserted). The poll is completed by reading the command pass through register to obtain the parallel poll response bits, then sending rpp with the C/S bit equal to zero. **Note that the IEEE 488 Standard requires a minimum of 2 microseconds before the parallel response is output to the bus.**

Send Interface Clear (sic) 0/1XX01111

This command is used when the zSBX 20 is system controller only. The Interface Clear (IFC) line is set true when this command is sent with C/S equal to 1. This must only be sent by the system controller and reset C/S equal to 0 after the IEEE 488 Standard minimum time for IFC has elapsed (100us.). **A longer time of about 5 milliseconds is suggested.** The system controller is put into the controller active state and a BO interrupt is generated (if enabled).

Send Remote Enable (sre) 0/1XX10000

This command instructs the zSBX 20 to set the REN line true thus sending the remote enable message over the GPIB. REN is set false by sending sre with C/S equal to zero, causing the GPIB devices to return to local mode. This command is used only when the zSBX 20 is the system controller.

Request Control (rqc) naXX10001

Multiple GPIB controllers are allowed on the bus although only one controller can be actively in control at one time. Also, there can only be one ultimate "system" controller. The function of the system controller is to take control at power-up time and during system conflicts. Only the system controller is allowed to assert Interface Clear (IFC) and Remote Enable (REN).

A controller may "pass control" to another controller via the GPIB message Take Control (TCT). The current controller-in-charge passes control to the zSBX 20 by sending the zSBX 20 talk address followed by the TCT message. The zSBX 20 recognizes the TCT by receiving an Unrecognized Command Group (UNC) interrupt, if enabled, and reading the TCT message from the Command Pass-Through Register. The zSBX 20 responds to TCT by issuing the rqc command. The GPIB hardware then waits for the current controller-in-charge to release Attention and then asserts Attention itself, going into the controller Active State. A BO interrupt is generated if enabled.

Release Control (rlc) naXX10010

The zSBX 20 may pass control (or return control) to another controller by the same protocol as the rqc discussion mentioned previously. In this case, TCT is sent by the zSBX 20, following the "new" controller talk address. After the handshake is completed, the zSBX 20 issues the rlc auxiliary command which releases the Attention line, thus relinquishing control. The "new" controller must then take (retake) control.

Note to users: There is no standard protocol to enable a controller to regain control once it has passed control to another device. The system designer must provide a way to alert potential controllers and current controllers that a transfer of control needs to take place. An easy method to accomplish this is through the use of a serial poll protocol. This way, an inactive controller can request service of the current active controller via the SRQ line. When the inactive controller sends its serial poll response byte during the serial poll, the response byte must contain the information indicating a request for control. If more than two controllers are used in a system, then the designer must assign a priority scheme should multiple controllers request control simultaneously.

Disable All Interrupts (dai) 0/1XX10011

This command disables the GPIB interrupt line. The interrupt registers and any holdoffs selected are not affected. This feature is useful in systems designed for polling operation as opposed to interrupt operation.

Pass Through Next Secondary (pts) naXX10100

This command may be used to carry out a remote configuration of a parallel poll. The parallel poll configure command (PPC) is passed through the zSBX 20 as an unrecognized command and must be identified by the CPU. The "pts" command is issued, and the next byte received by the zSBX 20 is passed through via the command pass-through register. This should be the parallel poll enable (PPE) message which is read by the microprocessor.

Set T1 Delay (stdl) 0/1XX10101

This command is used by the zSBX 20 to set the data to Data Valid delay time T1. The T1 delay time will be set to 6 clock cycles (1.2 microseconds at 5 MHz) if this command is sent with the C/S bit at one. The T1 delay time is 10 clock cycles (2 microseconds at 5 MHz) following a power on RESET, or if stdl is sent with C/S set to zero.

Three-state driver mode is required when using the short T1 time, to reduce the settling time of data on the DIO lines. See the GPIB driver discussion in this section.

Shadow Handshake (shdw) 0/1XX10110

This auxiliary command enables the controller to carry out the listener handshake without participating in a data transfer. The Data Accepted line (DAC) is pulled true a maximum of 3 clock cycles after Data Valid (DAV) is received. Not Ready For Data (NRFD) is allowed to go false as soon as DAV is removed. It must be used in conjunction with the "lon" mode. The END interrupt can also indicate when to generate an ACDS holdoff. This permits the controller to sense the end of string transfers across the GPIB.

The shadow handshake function allows the "tcs" command to be synchronized with the Acceptor Not Ready State (ANRS) so that ATN can be re-asserted without causing the loss or corruption of a data byte. The END interrupt can also be received to cause a RFD holdoff to be generated.

Set Very Fast T1 Delay (vstdl) 0/1XX10111

The IEEE 488 specification allows the bus settling time "T1" to be reduced to 400 NS on all bytes except the first byte after Attention (ATN) is unasserted. Then it is to be greater than 1100 NS. The zSBX 20 has a feature which reduces "T1" to 600 NS (3 clock cycles at 5 MHz) on all bytes but the first when ATN is unasserted. When ATN is unasserted or on the first byte, "T1" will be 2 microseconds with stdl not set or 1.2 microseconds with "stdl" set. The feature is programmable. The vstdl is a clear/set type command. If vstdl is set, three-state drivers are required for shorter settling time for data.

Second Service Request (rsv2) 0/1XX11000

This auxiliary command should be used by a device to request service from a GPIB controller. Once set true, and a SPAS interrupt occurs (indicating that the serial poll response byte has been read), this bit is automatically reset by the TMS 9914A logic. Most systems should request service with this command as opposed to RSV1 in the Serial Poll Register. Please refer to the serial poll register description for more information.

GPIB TRANSCEIVERS (7516/0,2)

The 7516/0,2 GPIB transceiver chips ensure that all relevant bus driver/receiver specifications are met. These transceivers feature:

* 500 mV Receiver Hysteresis
* Bus-Terminating Resistors
* No loading with no power
* Meets IEEE 488 - 1978 Standard

The sixteen signal lines required by the interface system are implemented with two devices. The 75160A handles the 8-bit data bus and the 75162A handles the handshake lines and bus management signals.

The 75160A has a Pull-up Enable pin (PE) that controls the output characteristics. When PE is low, the 75160A has the characteristics of open-collector outputs. When PE is high, three-state characteristics are exhibited. The state of this line is normally three-state, except during parallel polls.

The 75162A octal bus transceiver determines the direction of REN and IFC via the system controller (SC) input. This input is connected to a DIP switch at location 1A, position 8, and is shipped from the factory enabled. In the enabled mode, the zSBX 20 acts as a system controller by sending Remote/Local and Interface Clear messages.

Note: Older 7516/0,2 chips glitch the GPIB lines when powered on or off.

The GPIB system commands to be discussed are shown in Figure 18.

INIT	INITIALIZATION
Talker/Listener	
SEND	SEND DATA
RECV	RECEIVE DATA
XFER	TRANSFER DATA
Controller	
TRIG	GROUP EXECUTE TRIGGER
DCLR	DEVICE CLEAR
SPOL	SERIAL POLL
PPEN	PARALLEL POLL ENABLE
PPDS	PARALLEL POLL DISABLE
PPUN	PARALLEL POLL UNCNFG.
PPOL	PARALLEL POLL
PCTL	PASS CONTROL
RCTL	RECEIVE CONTROL
SCND	SEND COMMAND STRING
SRQD	SERVICE REQUESTED
System Controller	
REME	REMOTE ENABLE
LOCL	LOCAL
IFCL	ABORT/INTERFACE CLEAR

Figure 18. GPIB System Commands.

Each system command discussed is implemented as a subroutine. Each subroutine documents the necessary parameters that must be passed to it to operate properly. These subroutines, or drivers, assume that only primary addresses will be used on the GPIB. The zSBX 20 GPIB Primary address is read from the DIP switch located at pack position 3A. The five least significant switches (switches 1-5) are read and sent to the GPIB by the subroutines. To use secondary addresses, the test for valid talker/listener address (range macro) must be modified to include secondaries. Also assumed is that the controller is the system controller.

INITIALIZATION

Initialization

This routine is called after power-on or after a hardware reset before any GPIB activity occurs. Interface Clear (IFC) is sent for approximately 5 ms. All interrupts are disabled and the fast GPIB data settling rate (T1) is set (note: GPIB transceivers must be in three-state mode).

```
INIT:                                   ;Initialize 9914
   Reset                                ;Send reset to 9914
   IFC                                  ;Output IFC for 5ms.
   All interrupts off
   Set fast T1                          ;For 3-state drivers
   ton                                  ;Set talk only
   Return
```

Figure 19. GPIB Controller Initialization (INIT).

TALKER/LISTENER ROUTINES

Send Data

SEND <listener list pointer><count>
<EOS><data buffer pointer>

This system command sends data from the CPU to one or more devices. The data is usually a string of ASCII characters, but may be binary or other forms as well. The data is device-specific.

My Talk Address (MTA) must be output to satisfy the GPIB requirement of only one talker at a time (any other talker will stop when MTA goes out).

This routine assumes a non-null listener list in that it always sends Universal Unlisten. If it is desired to send data to the listeners previously addressed, one could add a check for a null list and not send UNL. Count must be 64k or less due to a 16 bit register. This routine always uses count or an EOS character to terminate data tranmission. EOI is sent with the last byte. See Figure 20.

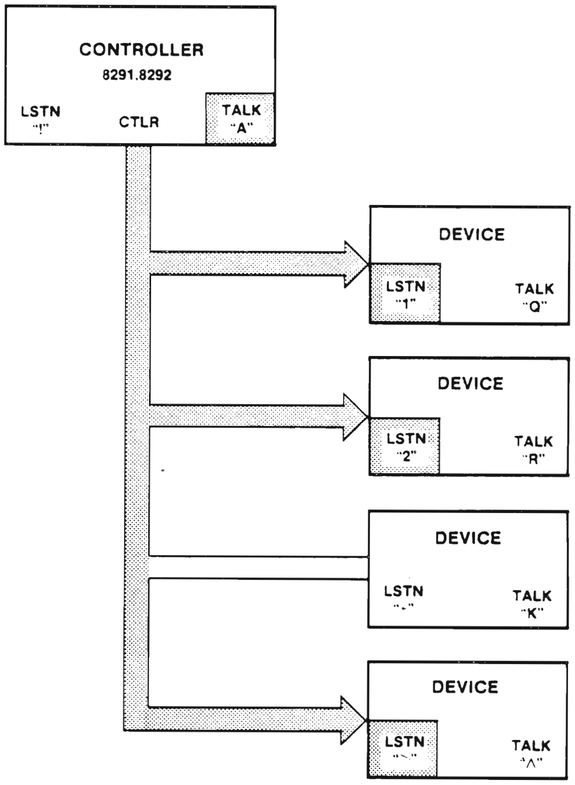


Figure 20. SEND to "1","2", ">"; "ABCD";EOS="D".

```
SEND:                                  ;Send non-DMA
   MTA, UNL                            ;We will talk, nobody listen
   While 20H < listener < 3EH          ;GPIB listen addresses are
      output-to-9914 listener          ;"space" thru ">" ASCII
      Increment listen list pointer    ;Address all listeners
   Output-to-9914 GTS                  ;stops asserting ATN, go
                                       ;to standby
   If count < > 0 then
      If count=1 or EOS then FEOI      ;Send EOI with last byte
      Decrement count
      Increment data pointer
      Output-to-9914 data              ;output GPIB data
      Repeat until all done
   Output-to-9914 TCA                  ;assert ATN, take cont. sync.
   Return
```

Figure 21. GPIB Controller Send (SEND).

Receive Data

RECV <talker><count><EOS>
<data buffer pointer>

This system command is used to input data from a device. The data is typically a string of ASCII characters.

This routine is the dual of SEND. It assumes a new talker will be specified, a count of less than 64k, and an EOS character or count to terminate the input. My Listen Address (MLA) is sent to keep the GPIB transactions totally regular to facilitate analysis by a GPIB logic analyzer like the Ziatech ZT 488. Otherwise, the bus would appear to have no listener even though the zSBX 20 will be listening.

Note that although the count may go to zero before the transmission ends, the talker will probably be left in a strange state and may have to be cleared by the controller. The count ending of RECV is therefore used as an error condition in most situations.

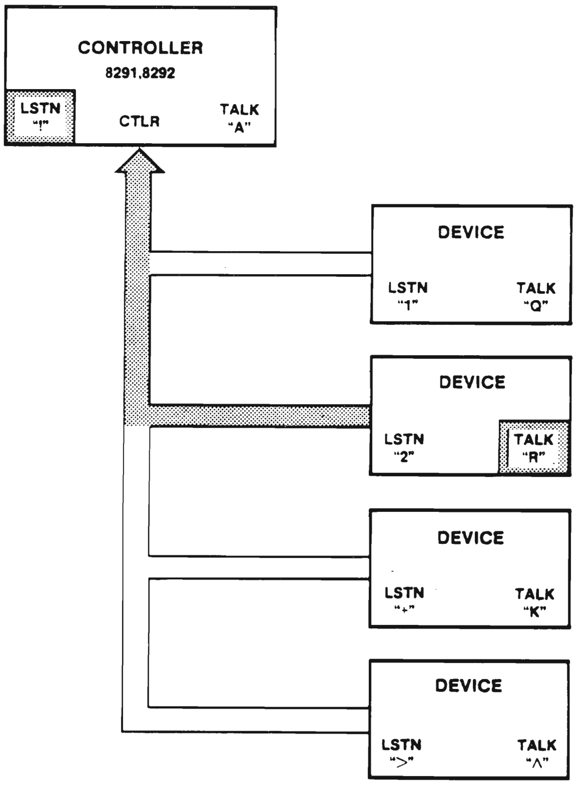


Figure 22. RECV from "R";EOS=0DH.

```
RECV:                                  ;Receive data non-DMA
   If 40H < talker < 5EH then          ;GPIB talk addresses are
      Output-to-9914 talker            ;"@" thru "" ASCII
   Increment talker pointer
   Output-to-9914 UNL, MLA             ;Everyone except us stop
                                       ;listening
   Enable-9914
      Holdoff on all data              ;Stop, check for EOS character
      lon, reset ton                   ;Listen only (no talk)
   Output-to-9914 GTS                  ;9914 stops asserting ATN, go
                                       ;to standby
      Input-from-9914 data             ;input data, byte by byte
      Loop till done
   Output-to-9914 TCS                  ;9914 asserts ATN take control
   Enable-9914                         ;Put 9914 back as before
      No holdoff on all data
      ton, reset lon
      Finish handshake                 ;Complete holdoff due to end,
                                       ;if any
      Return
```

Figure 23. GPIB Controller Receive (RECV).

Transfer Data

XFER<Talker><Listener list><EOS>

This system command is used to transfer data from a talker to one or more listeners where the controller does not participate in the transfer of the ASCII data. Thus the GPIB data rate is a function of the talker's transmitting speed and the listener's receiving speed. The controller will not receive any data, nor will it slow the transfer of data by handshaking each byte. This is accomplished through the shadow handshake mode while in listen-only.

This routine assumes a device list that has the ASCII talker address as the first byte and the string of (one or more) ASCII listener addresses following. The occurrence of EOI being sent by the talker will cause the controller to take control synchronously and thereby terminate the transfer.

Note: The talker must send EOI with the last byte for the controller to take charge.

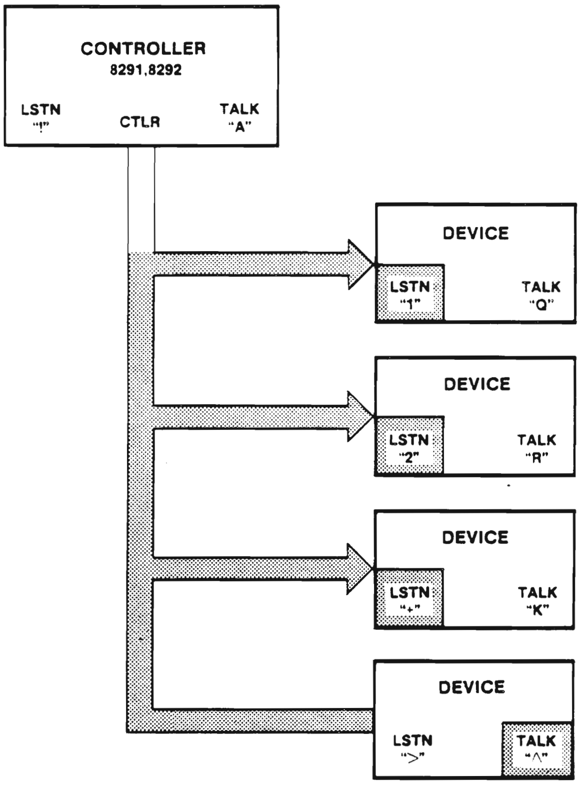


Figure 24. XFER from "" to "1","2", "+";EOS=0DH

```
XFER:
   Output-to-9914: Talker, UNL        ;Send talk address and UNL
   While 20H < listen < 3EH
     Output-to-9914: Listener         ;Send listen address
     Increment listen list pointer
   Enable-9914
     lon, no ton                      ;Controller is pseudo listener
     Shadow handshake                 ;Handshake but don't capture
                                      ;data
     Holdoff on EOI received          ;Capture EOI
   Output-to-9914: GTS                ;Go to standby
                                      ;9914 waits for EOI and then
     Take control synchronously       ;Regains control
   Enable 9914                        ;Go to Ready for Data
     Finish handshake
     Not shadow handshake
     Not holdoff on EOI
     ton
   Return
```

Figure 25. GPIB Controller Transfer (XFER).

CONTROLLER

Group Execute Trigger

TRIG<Listener List>

This system command causes a group execute trigger (GET) to be sent to all devices on the listener list. The intended use is to synchronize a number of instruments.

Device Clear

DCLR<Listener list>

This system command causes a Selective Device Clear (SDC) to be sent to all devices on the listener list. Note that this is not intended to clear the GPIB interface of the device, but should clear the device-specific logic.

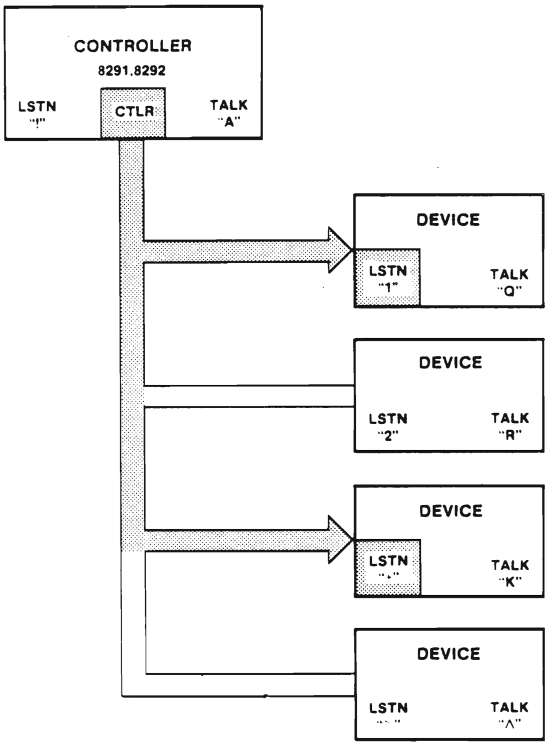


Figure 26. TRIG "1", "+".

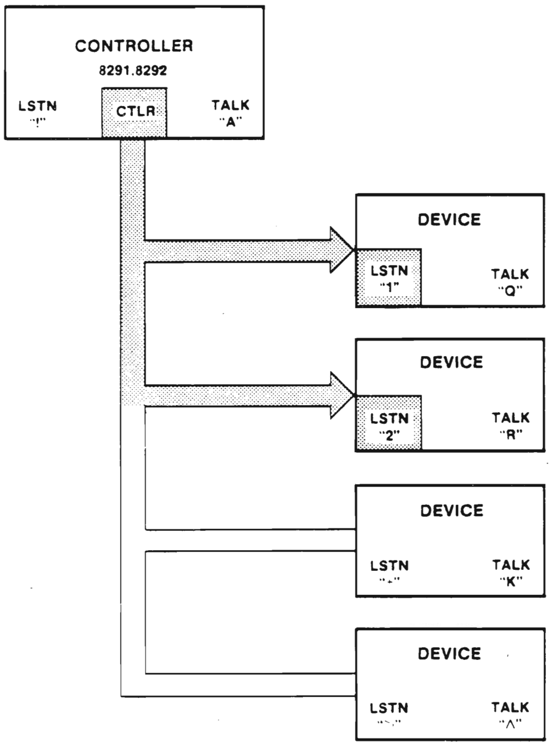


Figure 27. DCLR "1", "2".

```
TRIG:
   Output to 9914 UNL                  ;Everybody stop listening
   Check 20H<listener<3EH              ;Check listen address
      Output-to-9914 Listener          ;Address each listener
      Increment list pointer           ;Terminate on any bad listen
                                       ;address
   Output-to-9914 GET                  ;Issue group execute trigger
   Return
```

Figure 28. GPIB Controller Trigger (TRIG).

```
DCLR:
   Output-to-9914 UNL                      ;Everybody stop listening
   While 20H < Listener < 3EH              ;Check for valid listen addr.
     Output-to-9914 listener               ;Address each listener
     Increment listen list pointer         ;Terminate on any non-valid
                                           ;character
   Output-to-9914 SDC                      ;Selective device clear
   Return
```

Figure 29. GPIB Controller Device Clear (DCLR).

Serial Poll

SPOL<Talker list><status buffer pointer>

This system command sequentially addresses the designated devices and receives one byte of status from each. The bytes are stored in the buffer in the same order as the devices appear on the talker list. MLA is output for completeness.

A positive serial poll response (i.e., which device is requesting service) is determined by sequentially checking each response byte in the buffer to see if the second most significant bit is active.

Parallel Poll Enable

PPEN<Listener list><Configuration Buffer pointer>

This system command configures one or more devices to respond to Parallel Poll, assuming they implement subset PP1. The configuration information is stored in a buffer with one byte per device in the same order as devices appear on the listener list. The configuration byte has the format XXXXIP3P2P1 as defined by the IEEE Std. P3P2P1 indicates the bit # to be used for a response and 1 indicates the assertion value. See Sec. 2.9.3.3 of the Std. for more details.

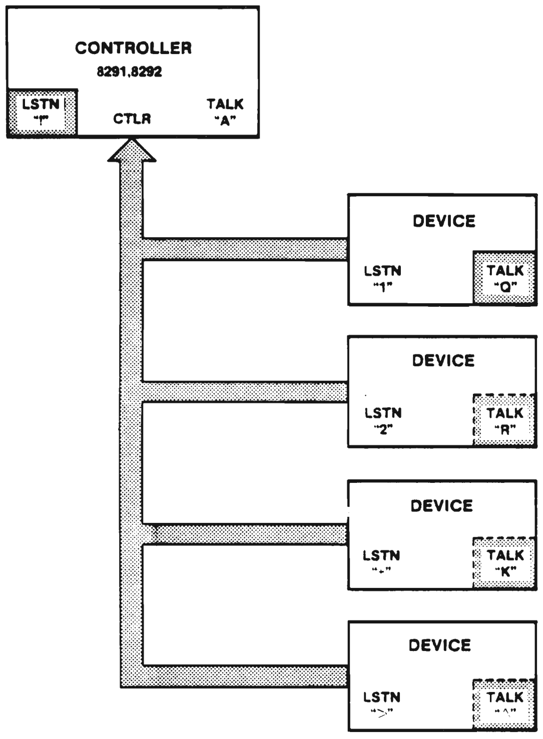


Figure 30. SPOL "Q", "R", "K", " "

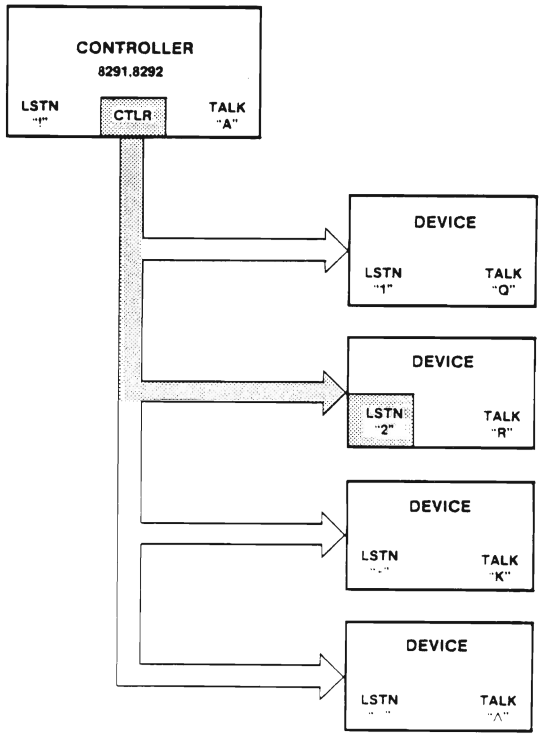


Figure 31. PPEN "2"; iP3P2P1 = 0111B.

```
SPOL:
   Output-to-9914 UNL, MLA, SPE      ;Unlisten, we listen, serial
                                     ;poll enable
                                     ;Only one byte of serial poll
                                     ;status from each talker
   While 40H<talker<5EH              ;Check for valid transfer
     Output-to-9914 talker           ;Address each device to talk
     Increment talker list pointer   ;One at a time
     Enable 9914
       lon, reset ton                ;Listen only to get status
       Holdoff on all
     Output-to-9914 GTS              ;Go to standby
     Wait for data in (BI)           ;Serial poll status byte
     Output-to-9914 TCS              ;Take control synchronously
     Input-from-9914 data            ;Actually get data from 9914
     Increment buffer pointer
     Enable 9914
       ton, reset lon
       No holdoff on all
   Output-to-9914 SPD                ;Send serial poll disable
                                     ;after all devices polled
   Return
```

Figure 32. GPIB Controller Serial Poll (SPOL).

```
PPEN:
   Output-to-9914 UNL                ;Universal unlisten
   While 20H<Listener<3EH            ;Check for valid listener
     Output-to-9914 listener         ;Stop old listener,
                                     ;address new
     Output 9914 PPC, (PPE+data)     ;Send parallel poll info
     Inc.listener list pointer       ;Point to next listener
     Increment buffer pointer        ;One configuration byte
                                     ;per listener
     Repeat for each listener        ;Finish listen list
   Return
```

Figure 33. GPIB Controller Parallel Poll Enable (PPEN).

Parallel Poll Disable

PPDS<listener list>

This system command disables one or more devices from responding to a Parallel Poll by issuing a Parallel Poll Disable (PPD). It does not deconfigure the devices.

Parallel Poll Unconfigure

PPUN

This system command deconfigures the Parallel Poll response of all devices by issuing a Parallel Poll Unconfigure message.

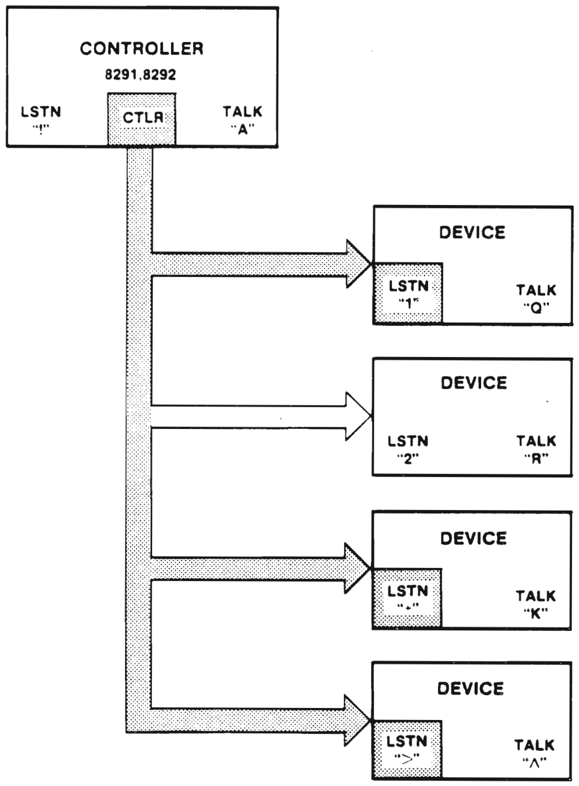


Figure 34. PPDS "1", "+", ">".

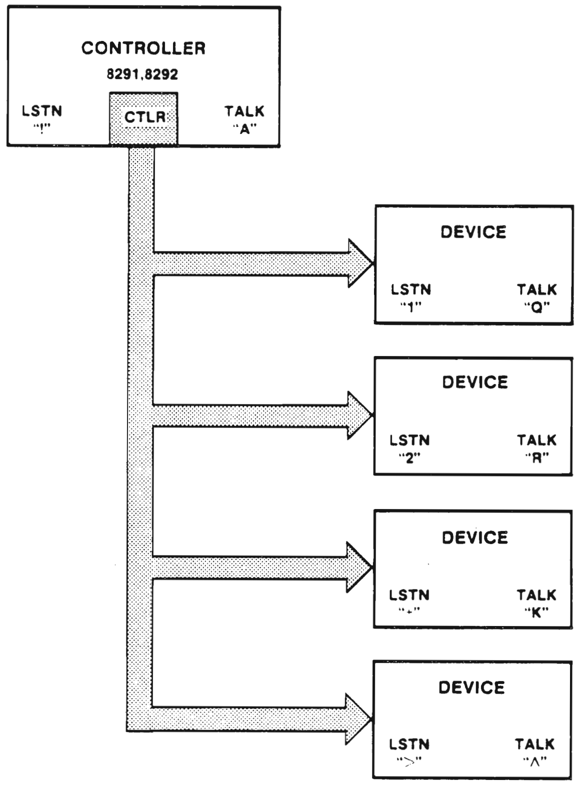


Figure 35. PPUN

```
PPDS:
   Output-to-9914 UNL               ;Universal Unlisten
   While 20H<Listener<3EH           ;Check for valid listener
     Output-to-9914 listener        ;Address listener
     Increment listener list pointer
   Output-to-9914 PPC, PPD          ;Disable PP on all listeners
   Return
```

Figure 36. GPIB Controller Parallel Poll Disable (PPDS).

```
PPUN:
   Output-to-9914 PPU              ;Unconfigure all parallel poll
   Return
```

Figure 37. GPIB Controller Parallel Poll Unconfigure (PPUN).

```
PPOL:
   Output-to-9914 RPP              ;Execute parallel poll
   Return Data (status byte)       ;From Command Pass Thr.
```

Figure 38. GPIB Controller Conduct Parallel Poll (PPOL).

Conduct a Parallel Poll

PPOL

This system command causes the controller to conduct a Parallel Poll on the GPIB for approximately 12.5 ms. (at 6 MHz). Note that a parallel poll does not use the handshake; therefore, to ensure that the device knows whether or not its positive response was observed by the controller, the CPU should explicitly acknowledge each device by a device-dependent data string. Otherwise, the response bit will still be set when the next Parallel Poll occurs.

Pass Control

PCTL<talker>

This system command allows the controller to relinquish active control of the GPIB to another controller. Normally some software protocol should already have informed the controller to expect this, and defined under what conditions to return control. The TMS 9914 must be set up to become a normal device and the CPU must handle all commands passed through, otherwise control cannot be returned (see Receive Control below). The controller will go idle.

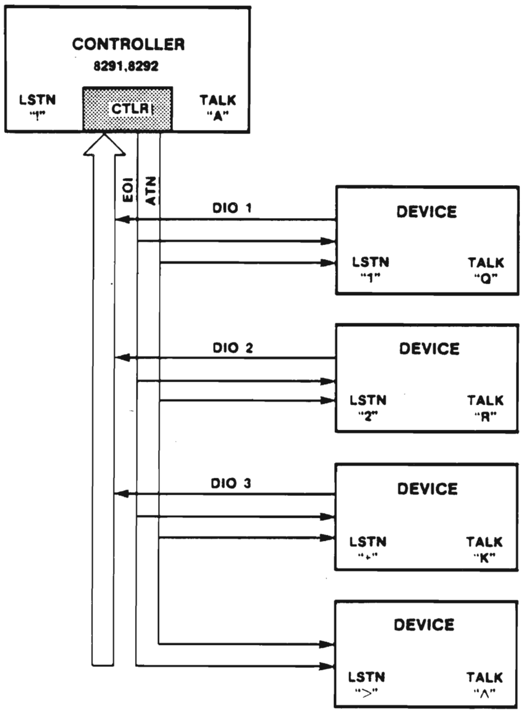


Figure 39. PPOL

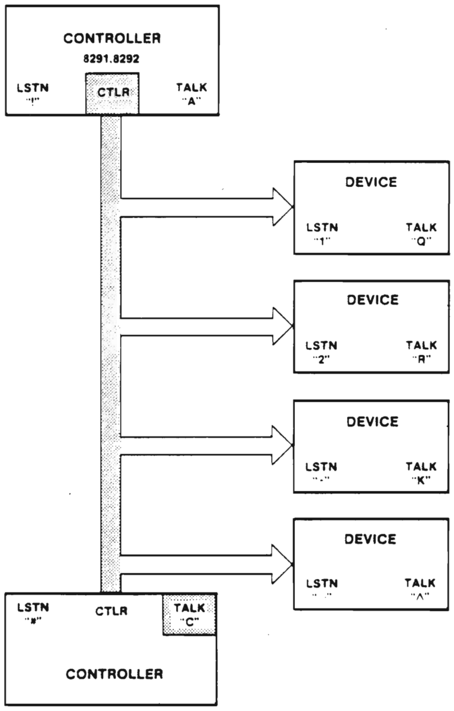


Figure 40. PCTL "C"

```
PCTL:
  If 40H<talker<5EH then
    if talker< > MTA then            ;Cannot pass control to myself
      output-to-9914 talker, TCT     ;Take cntrl. message to talker
      Enable-9914                    ;Set up 9914 as normal device
        not ton, not lon
        My device address            ;Put device number in
                                     ;address register
        Undefined comd pass thr.     ;Required to receive control
   Output-to-9914 RLCT               ;Put controller in idle
  Return
```

Figure 41. GPIB Controller Pass Control (PCTL).

Receive Control

RCTL

This system command allows this device to receive control from the current controller. This controller will receive control after being addressed to talk and receiving the Take Control Command.

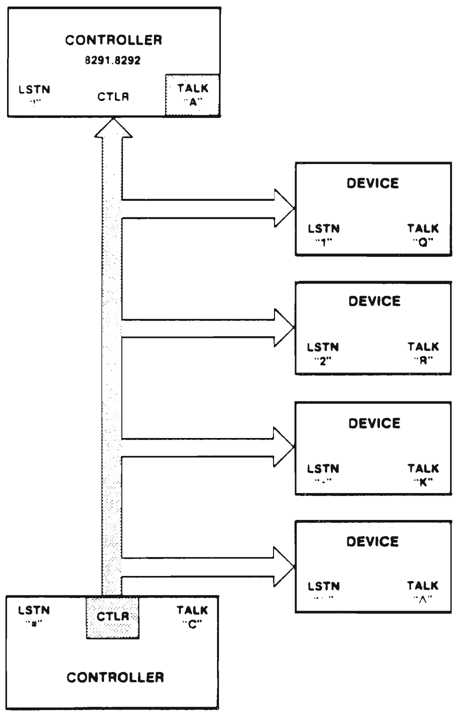


Figure 42. RCTL

Send Command String

SCND<command list, list end>

This system command is used to send a command string to the GPIB. A GPIB command is defined as having the attention (ATN) line asserted. The command string must be terminated by a list end character to insure proper operation. This command must be used with caution! Invalid commands can be easily generated.

Service Requested

SRQD

This system command is used to detect the occurrence of a Service Request on the GPIB. One or more devices may assert SRQ simultaneously, and the CPU would normally conduct a Serial Poll after calling this routine to determine which devices are SRQing.

```
RCTL:
  If Unidentified Command
    If TCT and TA                   ;Take control if Talk Addressed
       Clear INT Masks              ;Clear Interrupt Masks
       Output-to-9914 RQC           ;Request Control
       Set Accumulator > 0          ;Return Status
  Return
```

Figure 43. GPIB Controller Receive Control (RCTL).

```
SCMD:
  Get Command                       ;Get first command
    If not LEND                     ;Check for list end
       output-to-9914 CMD           ;No, output command
       increment pointer            ;Next command
       loop until done              ;Continue until done
  Return
```

Figure 44. GPIB Controller Send Command (SCND).

```
SRQD:
   If SRQ then                        ;Test status bit
     Return SRQ
   Else return no SRQ
```

Figure 45. GPIB Controller SRQ Occurred (SRQD).

SYSTEM CONTROLLER

Remote Enable

REME

This system command asserts the Remote Enable line (REN) on the GPIB. The devices will not go remote until they are later addressed to listen.

Local

LOCL

This system command deasserts the REN line on the GPIB. The devices will go local immediately. Depending Front panel controls on each instrument can now be activated by the user. This routine is the dual of Remote Enable routine (REME).

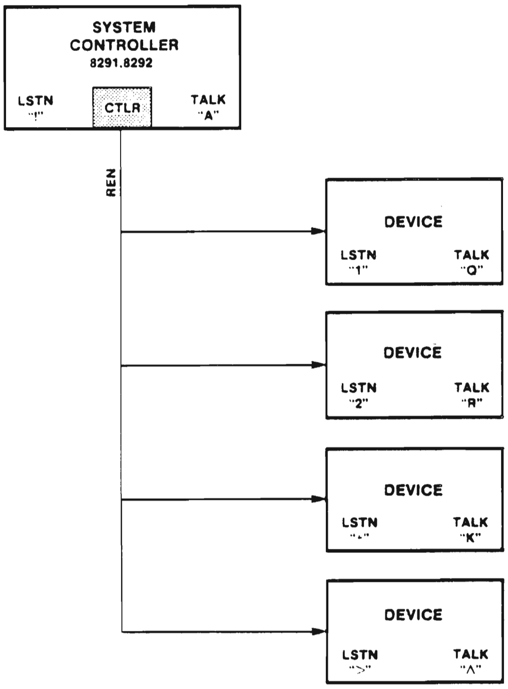


Figure 46. REME

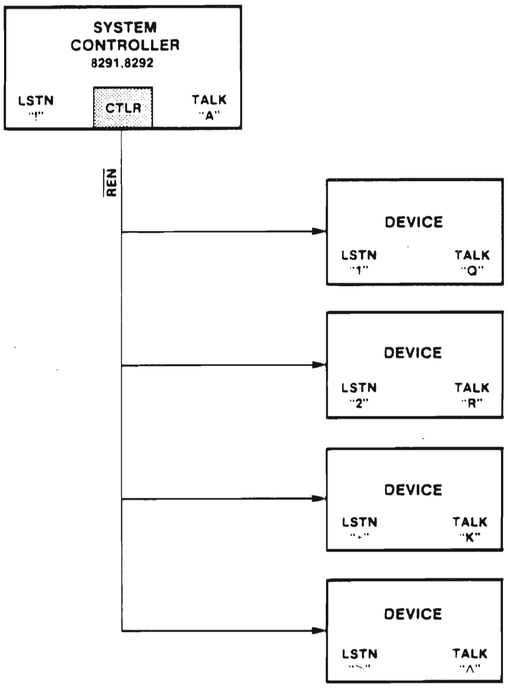


Figure 47. LOCL

```
REME:
   Output-to-9914 SRE               ;Assert remote enable line
   Return
```

Figure 48. GPIB Controller Remote Enable (REME).

```
LOCL:
   Output-to-9914 SRECLR            ;9914 stops asserting REM
   Return
```

Figure 49. GPIB Controller Return to Local (LOCL).

Interface Clear/Abort

IFCL

This system command asserts the GPIB's Interface Clear (IFC) line for at least 5 milliseconds. This causes all interface logic in all devices to go to a known state. Note that the device itself may or may not be reset, too. Most instruments do totally reset upon IFC. Some devices may require a DCLR as well as an IFCL to be completely reset. The (system) controller becomes Controller-in-Charge.

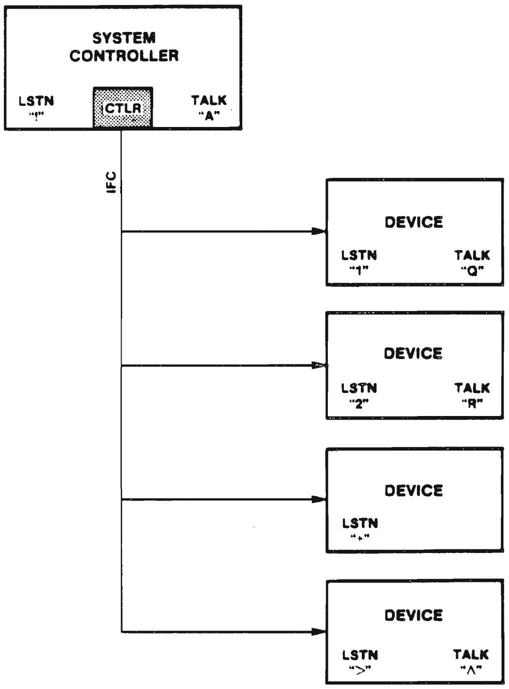


Figure 50. IFCL

```
IFCL:
   Output-to-9914 SIC                ;Assert Interface Clear
   Return                            ;For 5ms.
```

Figure 51. GPIB Controller Interface Clear (IFCL).

In a typical GPIB system where the zSBX 20 is a device, these subroutines can be used to send or receive data as addressed by the controller. Each subroutine documents the necessary parameters that must be passed to it in order to operate properly. To initiate operation the processor must load the parameters specified by each routine in certain registers and then use a "CALL" instruction to begin operation. Once data has been sent or received, a "RETURN" from the subroutine indicates completion.

When using the zSBX 20 as a device, a "CALL" to the Send or Receive routine is indicated by my (zSBX 20) address bit being set (please refer to the description on the interrupt status 1 register) and then waiting for LADS or TADS bit to be set. LADS would indicate a "CALL" to Receive, while TADS would indicate a "CALL" to Send.

No interrupts are used. To use interrupts, the TMS 9914A can be enabled to generate an interrupt on the occurrence of a byte input (BI) or a byte output (BO). Using interrupts would allow the CPU to execute other instructions while the GPIB sends or receives data.

INITIALIZATION

Initialization

This routine will initialize the TMS 9914A in the device mode with no interrupts enabled. The GPIB Device address is read from the DIP switch located at pack 3A. The five least significant switches (switches 1-5) are read and sent to the GPIB address register on the TMS 9914A. At power-on time or after a hardware reset, this routine must be executed before any GPIB activity is performed with the TMS 9914A used on the zSBX 20. Please refer to Figure 52.

TALKER/LISTENER ROUTINES

Send Data

SEND<count><EOS><data buffer pointer>

This routine sends data to one or more devices on the GPIB. The data is usually a string of ASCII characters, but may be binary or other forms as well. The data sent is thus device-specific.

Before initiating operation of this subroutine, My Talk Address (MTA) should have been received by the TMS 9914A. The address status of the TMS 9914A may be determined by the Talk Address Bit being set in the Address Status Register. Count must be 64K or less due to a 16 bit count register. The count or an EOS character is used to terminate the output string. In both cases EOI is sent with the last byte. Please refer to Figure 53.

Receive Data

This routine receives data from the current talker on the GPIB. The data is usually a string of ASCII characters, but may be one of several other forms as well (e.g., binary).

Before initiating operation of this subroutine My Listen Address (MLA) should have been received by the TMS 9914A. The address status of the TMS 9914A may be determined by the Listen Addressed bit being set in the Address Status Register.

When receiving data, count or EOS can terminate the input. Count must be less than 64K due to a 16 bit register.

Note that although the count may go to zero before the transmission ends, the talker will probably be left in a strange state and may have to be cleared by the controller. The count ending RECV is therefore used as an error condition in most situations (i.e., count is used as buffer limit protection, while EOI or EOS is typically used to terminate the input). Please refer to Figure 54.

```
INIT:                                   ;Initialize 9914
   Reset                                ;Send reset to 9914
   Set-up address                       ;Load my device
                                        ;Address in 9914
   Zero masks                           ;No interrupts
   Return
```

Figure 52. GPIB Device Initialization (INIT).

```
SEND:                                   ;Send non-DMA
   If count <>0 then
      If count=1 or EOS send EOI        ;Force EOI with last byte
      Decrement count
      Increment buffer pointer
      Output-to-9914 data               ;Send data to GPIB
      Repeat till all done              ;Repeat until done
   Return
```

Figure 53. GPIB Device Send (SEND).

```
RECV:                                   ;Receive data non-DMA
   Enable-9914
      Holdoff on all data               ;Stop, is it EOS Char.?
   If count <>0 then
      Input-from-9914 data              ;Input byte-by-byte
      Decrement count
      Increment data pointer
      If EOI or data=EOS                ;Input data until EOI or EOS
         Enable-9914
         No holdoff on all data         ;Put 9914 as before
         Finish handshake
   Return
```

Figure 54. GPIB Device Receive (RECV).

zSBX 20 JUMPER OPTIONS

The zSBX 20 has several user selectable options that can be jumper selected. Jumper selections are presented as a flow diagram with a detailed description to follow. Please refer to the photo of the zSBX 20 in Figure 55 for ease in locating the various jumpers.

Begin Jumper Selections.

If the zSBX 20's GPIB address is to be different than 4,
then set device address switch.

This DIP switch (pack position 3A) can be changed at any time to contain the GPIB device address. Please refer to the Address Switch Register description in the TMS 9914A I/O port description for more information. The GPIB address are set in switch numbers 0–4 requiring an AND with $1F to get the proper GPIB address.

If the zSBX 20 is a device only (i.e. not a controller),
then open the System Controller Switch.

This switch located at pack position 3A, #8 when opened disables the IFC ATN and REN lines from being asserted by the zSBX 20. The System Controller must now manage these lines.

Then set device address switch.
This DIP switch can be changed at any time to contain the GPIB device address. Please refer to the Addess Switch Register description in the TMS 9914A I/O port description for more information.

Then remove the shield ground.
Any bus system that connects many pieces of equipment together is a potential source of ground loops and spurious noise problems.

To help avoid these problems, only one GPIB device should connect the shield in the GPIB cable to earth ground. The ground is normally connected to ground by the System Controller. The shield connection may be removed via jumper W5 located next to the GPIB header.

If the zSBX 20 is to be port addressed by MCS1* rather than MCS0*,
then move W1 to W2.

The zSBX 20 requires 8 I/O port addresses to be accessed. MCS1* or MCS0* can be used to access the zSBX 20.

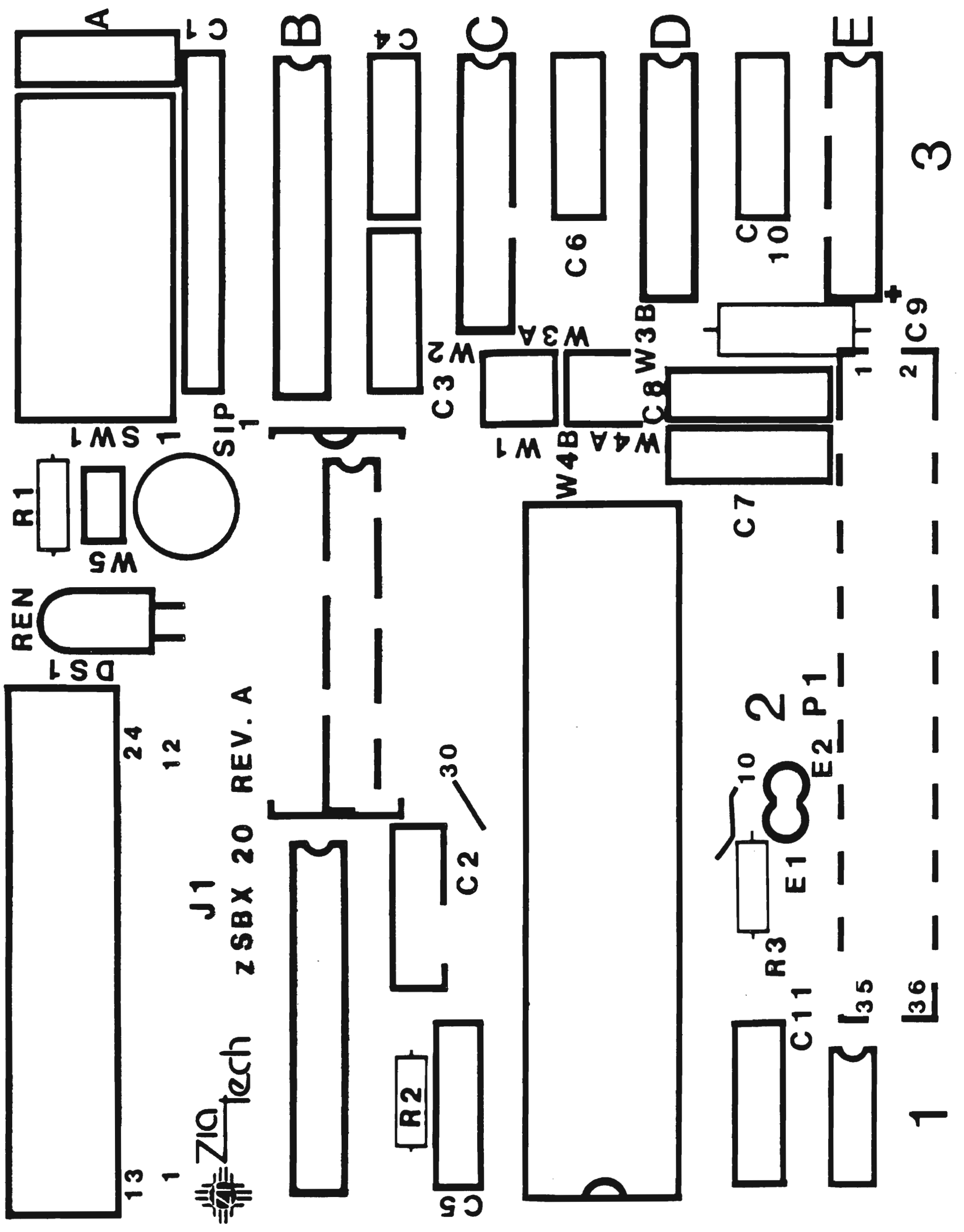
If an interrupt from the TMS 9914A is desired,
then install W4A for MINT0 or W4B for MINTR1.

The TMS 9914A can generate an interrupt on MINTR0 if the TMS 9914A is initialized with various interrupt masks.

If an interrupt from a trigger is desired,
then install W3B for MINTR0 or W3A for MINTR1.

The TMS 9914A can generate an interrupt with its trigger pin on MINTR1 if enabled.

End Jumper Selections.


A
B
C
D
E
C1
C4
C6
C10
C9
C3
C7
C8
C2
C5
C11
W1
W2
W3A
W3B
W4A
W4B
W5
SW1
SIP
1
R1
R2
R3
REN
DS1
J1
zSBX 20 REV. A
24
12
13
1
30
E1
E2
P1
10
35
36
1
2
3
zia tech

GPIB/IEEE 488 OVERVIEW

WHAT IS THE IEEE 488 (GPIB)?

The experience of designing systems for a variety of applications in the early 1970's caused Hewlett-Packard to define a standard intercommunication mechanism which would allow them to easily assemble instrumentation systems of varying degrees of complexity. In a typical situation each instrument designer designed his/her own interface from scratch. Each one was inconsistent in terms of electrical levels, pin-outs on a connector, and types of connectors. Every time they built a system they had to invent new cables and new documentation just to specify the cabling and interconnection procedures.

Based on this experience, Hewlett-Packard began to define a new interconnection scheme. They went further than that, however, for they wanted to specify the typical communication protocol for systems of instruments. So, in 1972, Hewlett-Packard came out with the first version of the bus which since has been modified and standardized by a committee of several manufacturers, coordinated through the IEEE, to perfect what is now known as the IEEE 488 Interface Bus (also known as the HP-IB, the GPIB and the IEC bus). While this bus specification may not be perfect, it is a good compromise of the various desires and goals of instrumentation and computer peripheral manufacturers to produce a common interconnection mechanism. It fits most instrumentation systems in use today and also fits very well the microcomputer I/O bus requirements. The basic design objectives for the GPIB were to:

1. Specify a system that is easy to use, but has all of the terminology and the definitions relating to that system precisely spelled out so that everyone uses the same language when discussing the GPIB.

2. Define all of the mechanical, electrical, and functional interface requirements of a system, yet not define any of the device aspects (they are left up to the instrument designer).

3. Permit a wide range of capabilities of instruments and computer peripherals to use a system simultaneously and not degrade each other's performance.

4. Allow different manufacturers' equipment to be connected together and work together on the same bus.

5. Define a system that is good for limited distance interconnections.

6. Define a system with minimum restrictions on performance of the devices.

7. Define a bus that allows asynchronous communication with a wide range of data rates.

8. Define a low cost system that does not require extensive and elaborate interface logic for the low cost instruments, yet provides higher capability for the higher cost instruments if desired.

9. Allow systems to exist that do not need a central controller; that is, communication directly from one instrument to another is possible.

Although the GPIB was originally designed for instrumentation systems, it became obvious that most of these systems would be controlled by a calculator or computer. With this in mind, several modifications were made to the original proposal before its final adoption as an international standard. The following list highlights the salient characteristics of the GPIB as both an instrumentation bus and as a computer I/O bus.

1. Data Rate
   1M bytes/s,max.
   250k bytes/s,typ.

2. Multiple Devices
   15 devices, max (elect. limit)
   8 devices, typ (interrupt flexibility)

3. Bus Length
   20 m, max.
   2 m/device, typ.

4. Byte Oriented
   8-bit commands
   8-bit data

5. Block Multiplexed
   Optimum strategy on GPIB due to setup overhead for commands

6. Interrupt Driven
   Serial poll (slower devices)
   Parallel poll (faster devices)

7. Direct Memory Access
   One DMA facility at controller serves all devices on bus

8. Asynchronous
   One talker
   Multiple listeners

9. I/O to I/O Transfers
   Talker and listeners need not include microcomputer/controller

The bus can be best understood by examining each of the above characteristics from the viewpoint of a general microcomputer I/O bus.

Data Rate -- Most microcomputer systems utilize peripherals of differing operational rates, such as floppy discs at 31k or 62k bytes/s (single or double density), tape cassettes at 5k to 10k bytes/s, and cartridge tapes at 40k to 80k bytes/s. In general, the only devices that need high speed I/O are 0.5" (1.3 cm) magnetic tapes and hard discs, operational at 30k to 781k bytes/s, respectively. Certainly the 250k bytes/s data rate that can be easily achieved by the IEEE 488 bus is sufficient for microcomputers and their peripherals, and is more than needed for typical analog instruments that take only a few readings per second. The 1M byte/s maximum data rate is not easily achieved on the GPIB and requires special attention to considerations beyond the scope of this note. Although not required, data buffering in each device will improve the overall bus performance and allow more utilization of the bus bandwidth.

Multiple Devices -- Many microcomputer systems used as computers (not as components) service from three to seven peripherals. With the GPIB, up to 8 devices can be handled easily by 1 controller; with some slowdown in interrupt handling, up to 15 devices can work together. The limit of 8 is imposed by the number of unique parallel poll responses available; the limit of 15 is set by the electrical drive characteristics of the bus. Logically, the IEEE 488 Standard is capable of accommodating more device addresses (31 primary, each potentially with 31 secondaries).

Bus Length -- Physically, the majority of microcomputer systems fit easily on a desk top or in a standard 19" (48 cm) rack, eliminating the need for extra long cables. The GPIB is designed typically to have 2 m of length per device, which accommodates most systems. A line printer might require greater cable lengths, but this can be handled by using extra dummy terminations. Overall bus length should be kept at a minimum to ensure data integrity.

Byte Oriented -- The 8-bit byte is almost universal in I/O applications; even 16-bit and 32-bit computers use byte transfers for most peripherals. The 8-bit byte matches the ASCII code for characters and is an integral submultiple of most computer word sizes. The GPIB has an 8-bit wide data path that may be used to transfer ASCII or binary data, as well as status and control bytes.

Block Multiplexed -- Many peripherals are block-oriented or are used in a block mode. Bytes are transferred in a fixed or variable length group; then there is a wait before another group is sent to that device, e.g., one sector of a floppy disc, one line on a printer or tape punch, etc. The GPIB is, by nature, a block-multiplexed bus due to the overhead involved in addressing various devices to talk and listen This overhead is less bothersome if it only occurs once for a large number of data bytes (once per block). This mode of operation matches the needs of microcomputers and most of their peripherals. Because of block multiplexing, the bus works best with buffered memory devices and/or devices which have the capability of operating with Direct Memory Access (DMA).

Interrupt Driven -- Many types of interrupt systems exist, ranging from complex, fast, vectored/priority networks to simple polling schemes. The main tradeoff is usually cost versus speed of response. The GPIB has two interrupt protocols to help span the range of applications. The first is a single service request (SRQ) line that may be asserted by all interrupting devices. The controller then polls all devices to find out which wants service. The polling mechanism is well defined and can be easily automated. For higher performance, the parallel poll capability in the IEEE 488 allows up to eight devices to be polled at once -- each device is assigned to one bit of the data bus. This mechanism provides fast recognition of an interrupting device. A drawback is the frequent need for the controller to explicitly conduct a parallel poll, since there is no equivalent of the SRQ line for this mode.

Direct Memory Access (DMA) — In many applications, no immediate processing of I/O data on a byte-by-byte basis is needed or wanted. In fact, programmed transfers slow down the data transfer rate unnecessarily in these cases, and higher speed can be obtained using DMA. With the GPIB, one DMA facility at the controller serves all devices. There is no need to incorporate complex logic in each device.

Asynchronous Transfers — An asynchronous bus is desirable so that each device can transfer at its own rate. However, there is still a strong motivation to buffer the data at each device when used in large systems in order to speed up the aggregate data rate on the bus by allowing each device to transfer at top speed. The GPIB is asynchronous and uses a special 3-wire handshake that allows data transfers from one talker to many listeners.

I/O To I/O Transfers -- In practice, I/O to I/O transfers are seldom done due to the need for processing data and changing formats or due to mismatched data rates. However, the GPIB can support this mode of operation where the microcomputer is neither the talker nor one of the listeners. In this mode of operation, the transfer rate is determined by the operational speed of the devices.

GPIB SIGNAL LINES

Data Bus

The lines DIO1 through DIO8 are used to transfer addresses, control information and data. The formats for addresses and control bytes are defined by the IEEE 488 standard. Data formats are undefined and may be ASCII (with or without parity) or binary. DIO1 is the Least Significant Bit (note that this will correspond to bit 0 on most computers).

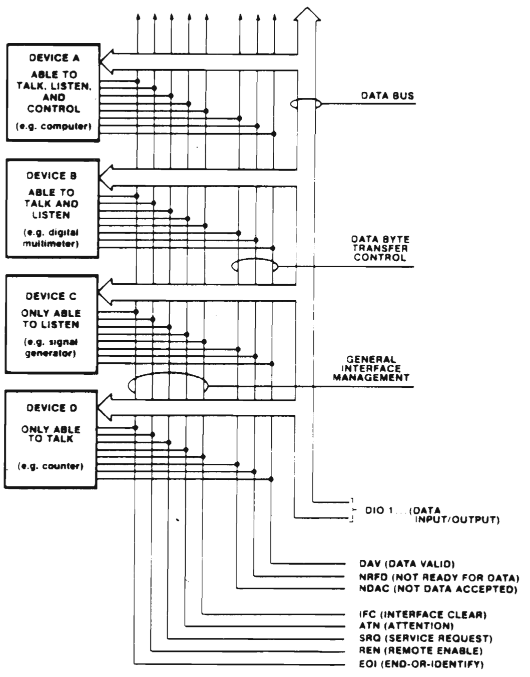


Figure 56. GPIB Interface Capabilities and Bus Structure.

Management Bus

ATN — Attention. This signal is asserted by the Controller to indicate that it is placing an address or control byte on the Data Bus. ATN is deasserted to allow the assigned Talker to place status or data on the Data Bus. The Controller regains control by reasserting ATN; this is normally done synchronously with the handshake to avoid confusion between control and data bytes.

EOI — End or Identify. This signal has two uses as its name implies. A talker may assert EOI simultaneously with the last byte of data to indicate end of data. The Controller may assert EOI along with ATN to initiate a Parallel Poll. Although many devices do not use Parallel Poll, all devices should use EOI to end transfers (many currently available ones do not).

SRQ — Service Request. This line is like an interrupt: it may be asserted by any device to request the Controller to take some action. The Controller must determine which device is asserting SRQ by conducting a Serial Poll at its earliest convenience. The requesting device deasserts SRQ when polled.

IFC — Interface Clear. This signal is asserted only by the System Controller in order to initialize all device interfaces to a known state. After deasserting IFC, the System Controller is the active controller of the system.

REN — Remote Enable. This signal is asserted only by the System Controller. Its assertion does not place devices into Remote Control mode; REN only enables a device to go remote when addressed to listen. When in Remote, a device should ignore its front panel controls.

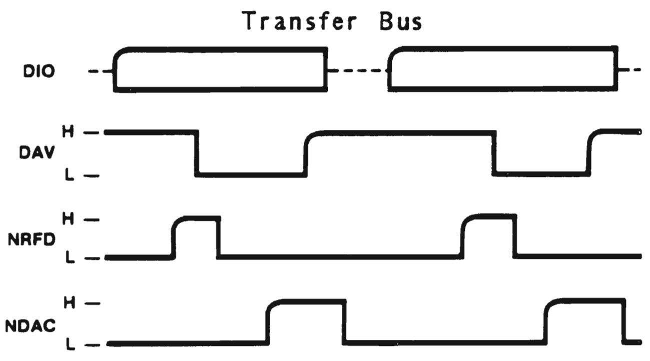


Figure 57. GPIB Handshake Sequence.

NRFD — Not Ready For Data. This handshake line is asserted by a listener to indicate it is not yet ready for the next data or control byte. Note that the Controller will not see NRFD deasserted (i.e. ready for data) until all devices have deasserted NRFD.

NDAC — Not Data Accepted. This handshake line is asserted by a Listener to indicate it has not yet accepted the data or control byte on the DIO lines. Note that the Controller will not see NDAC deasserted (i.e. data accepted) until all devices have deasserted NDAC.

DAV — Data Valid. This handshake line is asserted by the Talker to indicate that a data or control byte has been placed on the DIO lines and has had the minimum specified settling time.

GPIB INTERFACE FUNCTIONS

There are 10 interface functions specified by the IEEE 488 Standard. Not all devices will have all functions and some may only have partial subsets. The ten functions are summarized below with the relevant section number from the IEEE document given at the beginning of each paragraph. For further information please see the IEEE Standard.

1. SH — Source Handshake (section 2.3) This function provides a device with the ability to properly transfer data from a Talker to one or more Listeners using the three handshake lines.

2. AH — Acceptor Handshake (section 2.4) This function provides a device with the ability to properly receive data from the Talker using the three handshake lines. The AH function may also delay the beginning (NRFD) or end (NDAC) of any transfer.

3. T — Talker (section 2.5) This function allows a device to send status and data bytes when addressed to talk. An address consists of one (Primary) or two (Primary and Secondary) bytes. The latter is called an extended Talker.

4. L — Listener (section 2.6) This function allows a device to receive data when addressed to listen. There can be extended Listeners (analogous to extended Talkers above).

5. SR — Service Request (section 2.7) This function allows a device to request service from the Controller (interrupt). The SRQ line may be asserted asynchronously.

6. RL — Remote Local (section 2.8) This function allows a device to be operated in two modes: Remote via the GPIB or Local via the manual front panel controls.

7. PP — Parallel Poll (section 2.9) This function allows a device to present one bit of status to the Controller-in-charge. The device need not be addressed to talk and no handshake is required.

8. DC -- Device Clear (section 2.10) This function allows a device to be cleared (initialized) by the Controller. Note that there is a difference between the DC and the IFC lines.

9. DT — Device Trigger (section 2.11) This function allows a device to have its basic operation started either individually or as part of a group. This capability is often used to synchronize several instruments.

10. C -- Controller (section 2.12) This function allows a device to send addresses, as well as universal and addressed commands to other devices. There may be more than one controller on a system, but only one may be the controller-in-charge at any one time.

At power-on time the controller that is programmed to be the System Controller becomes the active controller-in-charge. The System Controller has several unique capabilities including the ability to send: (1) Interface Clear (IFC) - clears all device interfaces and returns control to the System Controller). (2) Remote Enable (REN) - allows devices to respond to bus data once they are addressed to listen) The System Controller may optionally Pass Control to another controller, if the system software has the capability to do so.

GPIB CONNECTOR

The GPIB connector is a standard 24-pin industrial connector such as Cinch or Amphenol series 57 Micro-Ribbon. The IEEE Standard specifies this connector, and the signal connections and the mounting hardware. The cable has 16 signal lines and 8 ground lines. The maximum length is 20 meters with no more than two meters per device (This last number is the average over all the bus).

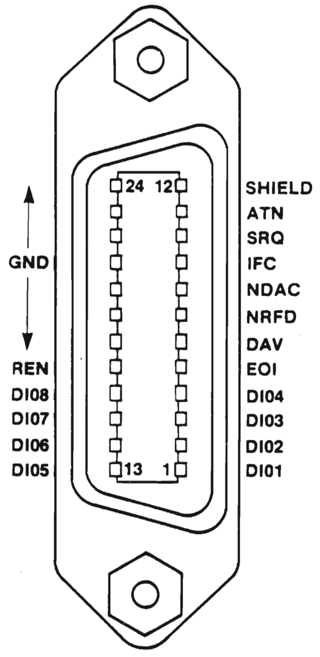


Figure 58. GPIB Connector.

GPIB SIGNAL LEVELS

The GPIB signals are all TTL compatible, low true signals. A signal is asserted (true) when its electrical voltage is less than 0.5 volts and is deasserted (false) when it is greater than 2.4 volts. Be careful not to become confused with the two handshake signals, NRFD and NDAC which are also low true (i.e., < 0.5 volts implies the device is Not Ready For Data).

GPIB MESSAGE PROTOCOLS

The GPIB is a very flexible communications medium and as such has many possible variations of protocols. To bring some order to the situation, this section will discuss a protocol similar to the one used by Ziatech's ZT 85 GPIB controller for Intel's MULTIBUS® computers.

DATA — Transfer a block of data from device A to devices B, C...

1. Device A Primary (Talk) Address
   Device A Secondary Address (if any)
2. Universal Unlisten
3. Device B Primary (Listen) Address
   Device B Secondary Address (if any)
   Device C Primary (Listen) Address

4. First Data Byte
   Second Data Byte
   .
   .
   Last Data Byte (EOI)
5. Null

TRIGR — Trigger devices A, B,...to take action
1. Universal Unlisten
2. Device A Primary (Listen) Address
   Device A Secondary Address (if any)
   Device B Primary (Listen) Address
   Device B Secondary Address (if any)
   etc.
3. Group Execute Trigger
4. Null

PSCTL — Pass control to device A
1. Device A Primary (Talk) Address
   Device A Secondary Address (if any)
2. Take Control
3. Null

CLEAR — Clear all devices
1. Device Clear
2. Null

REMAL — Remote Enable
1. Assert REN continuously

GOREM — Put devices A, B,...into Remote
1. Assert REN continuously
2. Device A Primary (Listen) Address
   Device A Secondary Address (if any)
   Device B Primary (Listen) Address
   Device B Secondary Address (if any)
   etc.
3. Null

GOLOC — Put devices A, B,...into Local
1. Universal Unlisten
2. Device A Primary (Listen) Address
   Device A Secondary Address (if any)
   Device B Primary (Listen) Address
   Device B Secondary Address (if any)
3. Go to Local
4. Null

LOCAL — Reset all devices to Local
1. Stop asserting REN

LLKAL — Prevent all devices from returning to local
1. Local Lock Out
2. Null

SPOLL — Conduct a serial poll of devices A, B,...
1. Universal Unlisten
2. Serial Poll Enable
3. ZT 85 Primary (Listen) Address
4. Device Primary (Talk) Address
   Device Secondary Address (if any)
5. Status byte from device
6. Go to Step 4 until all devices on list have been polled
7. Serial Poll Disable
8. Null

PPUAL — Unconfigure and disable Parallel Poll response from all devices
1. Parallel Poll Unconfigure
2. Null

ENAPP — Enable Parallel Poll response in devices A, B,...
1. Universal Unlisten
2. Device Primary (Listen) Address
   Device Secondary Address (if any)
3. Parallel Poll Configure
4. Parallel Poll Enable
5. Go to Step 2 until all devices on list have been configured.
6. Null

DISPP — Disable Parallel Poll response from devices A, B,...
1. Universal Unlisten
2. Device A Primary (Listen) Address
   Device A Secondary Address (if any)
   Device B Primary (Listen) Address
   Device B Secondary Address (if any)
   etc.
3. Parallel Poll Configure
4. Parallel Poll Disable
5. Null

DEBUGGING THE GPIB

The GPIB is a flexible interface that allows variations in protocol. A visual indication of GPIB activity from an inexpensive bus analyzer is very useful.

The ZT 488 is a cost effective GPIB analyzer. The ZT 488 monitors the bus while stepping through bus transactions one at a time. This allows the user to observe system operation, including device addressing, multiline commands, control lines and data.

The ZT 488 can simulate a missing device. This can help you check out the system software before the missing system device is installed.

The ZT 488 can also simulate a controller. This allows you to test a new device or debug a faulty device.

Three ZT 488 analyzer configurations are shown on the top of the next page. The configuration at the top shows the ZT 488 being used as a single-stepping monitor of a GPIB system. Next, the ZT 488 is shown emulating a GPIB "device". The bottom shows the ZT 488 emulating a GPIB controller.

The Ziatech Model 488 is the first truly low cost, handheld GPIB Analyzer. It conforms to the IEEE 488 specifications.

1 Easy to operate - automatic handshake or single step mode.

2 Compact - fits easily into a tool box, attache case, or on the lab bench.

3 Light Weight - under 1 pound.

4 Handheld, Portable - take it with you in the lab or in the field.

5 Low Cost - own your own, so you will have it when you need it most.

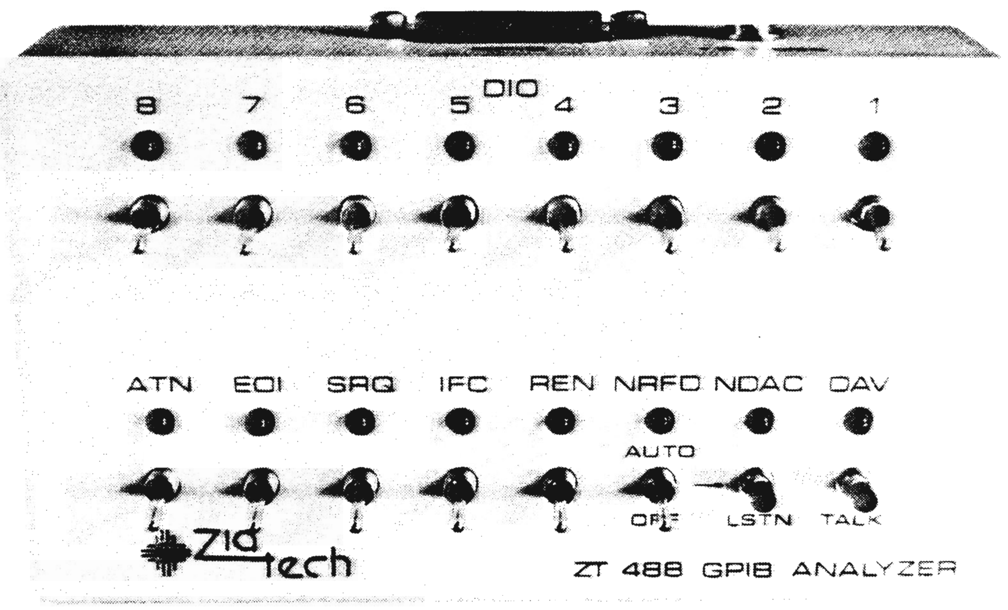
6 Reliable - all units burned in for 168 hours @ 55°C.

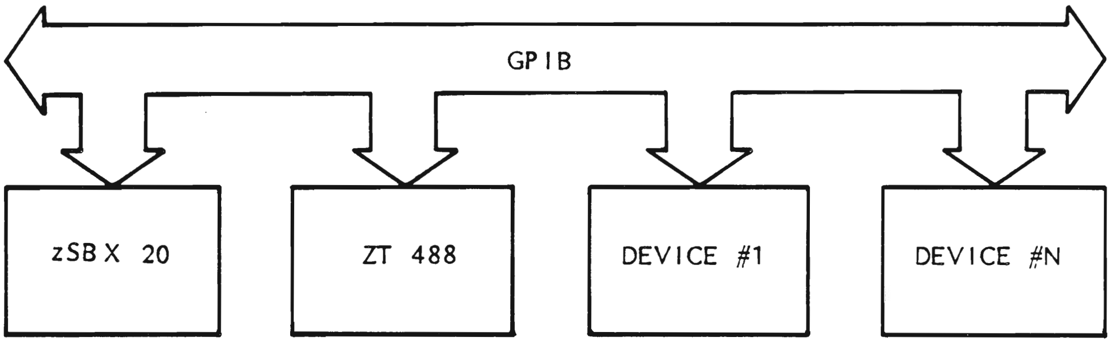
7 Easy Connection to System - uses standard metric thread GPIB connector.

8 Rugged - case made of high-impact polystyrene.

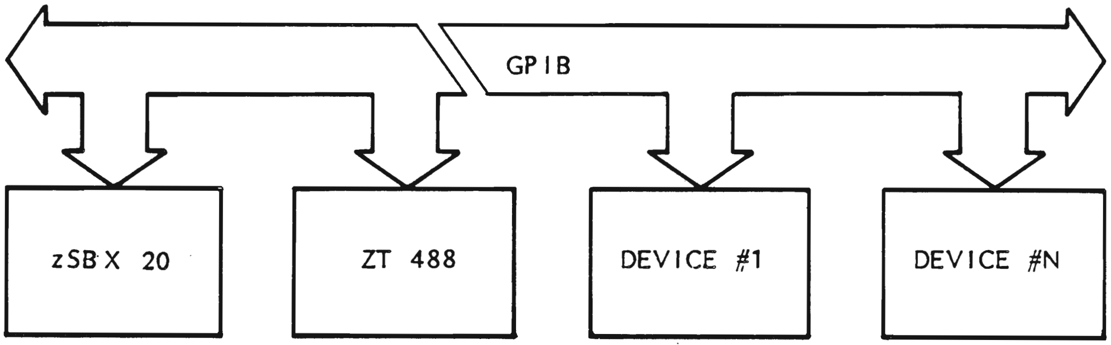
9 Low Power - 2 Watts @ 5VDC or 14 Watts @ 115VAC.

10 "Quick Reference" Back plate.

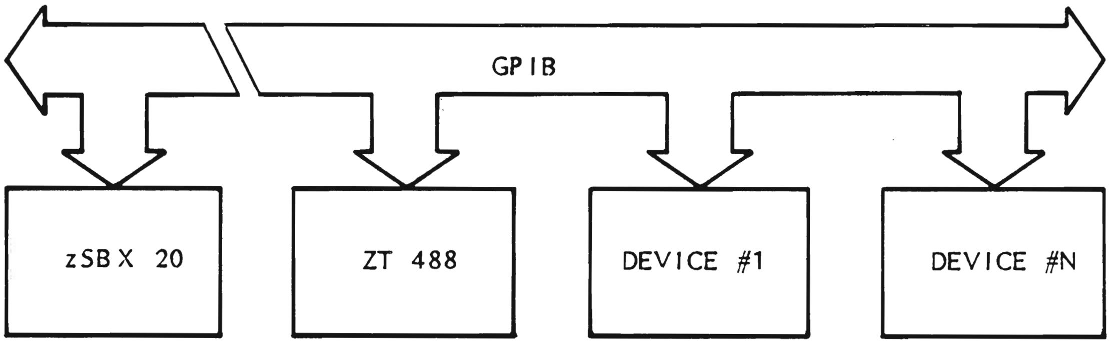




This configuration allows the ZT 488 to single step the entire GPIB, monitor the GPIB, or act as a controller or device. The analyzer may also request service from the controller.



This configuration allows the ZT 488 to analyze the GPIB controller by single-stepping the controller while checking address, data, and control signals.



This configuration allows the ZT 488 to analyze all GPIB devices by emulating the controller while checking the device's talk and listen features, along with serial and parallel polling.

GPIB REMOTE MESSAGE CODING

APPENDIX C

IEEE Standard 488-1978 lists all messages capable of being sent (talk) or received (listen) by an interface function, including both the encoding required to send the message and the decoding required to receive it. The logical state of each bus line signal is specified in the following message coding as 0, 1, Y or X, as follows:

0 = logical zero
1 = logical one
X = don't care (for received message)
Y = don't care (for send message)

Other symbols used in the remote message coding are:

U	= Uniline message
M	= Multiline message
AC	= Addressed Command
AD	= Address (talk or listen)
DD	= Device Dependent
HS	= Handshake
UC	= Universal Command
SE	= Secondary
ST	= Status

REMOTE MESSAGE CODING

					Bus Signal Line(s) and Coding That Asserts the True Value of the Message													
Mnemonic		Message Name	Type	Class	DIO 8	7	6	5	4	3	2	DIO 1	DAV NRFD NDAC	ATN	EOI	SRQ	IFC	REN
ACG	addressed command group		M	AC	Y	Ø	Ø	Ø	X	X	X	X	XXX	1	X	X	X	X
ATN	attention		U	UC	X	X	X	X	X	X	X	X	XXX	1	X	X	X	X
DAB	data byte	(Notes 1, 9)	M	DD	D8	D7	D6	D5	D4	D3	D2	D1	XXX	Ø	X	X	X	X
DAC	data accepted		U	HS	X	X	X	X	X	X	X	X	XXØ	X	X	X	X	X
DAV	data valid		U	HS	X	X	X	X	X	X	X	X	1XX	X	X	X	X	X
DCL	device clear		M	UC	Y	Ø	Ø	1	Ø	1	Ø	Ø	XXX	1	X	X	X	X
END	end		U	ST	X	X	X	X	X	X	X	X	XXX	Ø	1	X	X	X
EOS	end of string	(Notes 2, 9)	M	DD	E8	E7	E6	E5	E4	E3	E2	E1	XXX	Ø	X	X	X	X
GET	group execute trigger		M	AC	Y	Ø	Ø	Ø	1	Ø	Ø	Ø	XXX	1	X	X	X	X
GTL	go to local		M	AC	Y	Ø	Ø	Ø	Ø	Ø	Ø	1	XXX	1	X	X	X	X
IDY	identify		U	UC	X	X	X	X	X	X	X	X	XXX	X	1	X	X	X
IFC	interface clear		U	UC	X	X	X	X	X	X	X	X	XXX	X	X	X	1	X
LAG	listen address group		M	AD	Y	Ø	1	X	X	X	X	X	XXX	1	X	X	X	X
LLO	local lock out		M	UC	Y	Ø	Ø	1	Ø	Ø	Ø	1	XXX	1	X	X	X	X
MLA	my listen address	(Note 3)	M	AD	Y	Ø	1	L5	L4	L3	L2	L1	XXX	1	X	X	X	X
MTA	my talk address	(Note 4)	M	AD	Y	1	Ø	T5	T4	T3	T2	T1	XXX	1	X	X	X	X
MSA	my secondary address	(Note 5)	M	SE	Y	1	1	S5	S4	S3	S2	S1	XXX	1	X	X	X	X
NUL	null byte		M	DD	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	XXX	X	X	X	X	X
OSA	other secondary address		M	SE	(OSA = SCG ∧ $\overline{\text{MSA}}$)													
OTA	other talk address		M	AD	(OTA = TAG ∧ $\overline{\text{MTA}}$)													
PCG	primary command group		M	—	(PCG = ACG ∨ UCG ∨ LAG ∨ TAG)													
PPC	parallel poll configure		M	AC	Y	Ø	Ø	Ø	Ø	1	Ø	1	XXX	1	X	X	X	X
PPE	parallel poll enable	(Note 6)	M	SE	Y	1	1	Ø	S	P3	P2	P1	XXX	1	X	X	X	X
PPD	parallel poll disable	(Note 7)	M	SE	Y	1	1	1	D4	D3	D2	D1	XXX	1	X	X	X	X

REMOTE MESSAGE CODING (CONTINUED)

					Bus Signal Line(s) and Coding That Asserts the True Value of the Message													
Mnemonic	Message Name		Type	Class	DIO 8	7	6	5	4	3	2	DIO 1	DAV NRFD NDAC	ATN	EOI	SRQ	IFC	REN
PPR1	parallel poll response 1	(Note 10)	U	ST	X	X	X	X	X	X	X	1	XXX	1	1	X	X	X
PPR2	parallel poll response 2	(Note 10)	U	ST	X	X	X	X	X	X	1	X	XXX	1	1	X	X	X
PPR3	parallel poll response 3	(Note 10)	U	ST	X	X	X	X	X	1	X	X	XXX	1	1	X	X	X
PPR4	parallel poll response 4	(Note 10)	U	ST	X	X	X	X	1	X	X	X	XXX	1	1	X	X	X
PPR5	parallel poll response 5	(Note 10)	U	ST	X	X	X	1	X	X	X	X	XXX	1	1	X	X	X
PPR6	parallel poll response 6	(Note 10)	U	ST	X	X	1	X	X	X	X	X	XXX	1	1	X	X	X
PPR7	parallel poll response 7	(Note 10)	U	ST	X	1	X	X	X	X	X	X	XXX	1	1	X	X	X
PPR8	parallel poll response 8	(Note 10)	U	ST	1	X	X	X	X	X	X	X	XXX	1	1	X	X	X
PPU	parallel poll unconfigure		M	UC	Y	Ø	Ø	1	Ø	1	Ø	1	XXX	1	X	X	X	X
REN	remote enable		U	UC	X	X	X	X	X	X	X	X	XXX	X	X	X	X	1
RFD	ready for data		U	HS	X	X	X	X	X	X	X	X	XØX	X	X	X	X	X
RQS	request service	(Note 9)	U	ST	X	1	X	X	X	X	X	X	XXX	Ø	X	X	X	X
SCG	secondary command group		M	SE	Y	1	1	X	X	X	X	X	XXX	1	X	X	X	X
SDC	selected device clear		M	AC	Y	Ø	Ø	Ø	Ø	1	Ø	Ø	XXX	1	X	X	X	X
SPD	serial poll disable		M	UC	Y	Ø	Ø	1	1	Ø	Ø	1	XXX	1	X	X	X	X
SPE	serial poll enable		M	UC	Y	Ø	Ø	1	1	Ø	Ø	Ø	XXX	1	X	X	X	X
SRQ	service request		U	ST	X	X	X	X	X	X	X	X	XXX	X	X	1	X	X
STB	status byte	(Notes 8, 9)	M	ST	S8	X	S6	S5	S4	S3	S2	S1	XXX	Ø	X	X	X	X
TCT	take control		M	AC	Y	Ø	Ø	Ø	1	Ø	Ø	1	XXX	1	X	X	X	X
TAG	talk address group		M	AD	Y	1	Ø	X	X	X	X	X	XXX	1	X	X	X	X
UCG	universal command group		M	UC	Y	Ø	Ø	1	X	X	X	X	XXX	1	X	X	X	X
UNL	unlisten		M	AD	Y	Ø	1	1	1	1	1	1	XXX	1	X	X	X	X
UNT	untalk	(Note 11)	M	AD	Y	1	Ø	1	1	1	1	1	XXX	1	X	X	X	X

The 1/Ø coding on ATN when sent concurrent with multiline messages has been added to this revision for interpretive convenience.

NOTES:

(1) D1-D8 specify the device dependent data bits.

(2) E1-E8 specify the device dependent code used to indicate the EOS message.

(3) L1-L5 specify the device dependent bits of the device's listen address.

(4) T1-T5 specify the device dependent bits of the device's talk address.

(5) S1-S5 specify the device dependent bits of the device's secondary address.

(6) S specifies the sense of the PPR.

S	Response
Ø	Ø
1	1

P1-P3 specify the PPR message to be sent when a parallel poll is executed.

P3	P2	P1	PPR Message
Ø	Ø	Ø	PPR1
.	.	.	.
.	.	.	.
.	.	.	.
1	1	1	PPR8

(7) D1-D4 specify don't-care bits that shall not be decoded by the receiving device. It is recommended that all zeroes be sent.

(8) S1-S6, S8 specify the device dependent status. (DIO7 is used for the RQS message.)

(9) The source of the message on the ATN line is always the C function, whereas the messages on the DIO and EOI lines are enabled by the T function.

(10) The source of the messages on the ATN and EOI lines is always the C function, whereas the source of the messages on the DIO lines is always the PP function.

(11) This code is provided for system use, see 6.3.

The following pages illustrate example GPIB controller drivers for a board utilizing the TMS 9914A. This code is written in 8085 Assembly language and is provided as a model for writing your code. If you are using a different processor, this code will have to be modified for your application.

```
ASM8Ø :F1:85V1.SRC


ISIS-II 8Ø8Ø/8085 MACRO ASSEMBLER, V2.Ø        MODULE   PAGE    1
*** DRIVERS FOR ZT 85/38 CONTROLLER ***

  LOC  OBJ         SEQ         SOURCE STATEMENT

                     1 $TITLE ('*** DRIVERS FOR ZT 85/38 CONTROLLER ***')
                     2 $PAGEWIDTH(8Ø)
                     3 $MACROFILE
                     4 ;
                     5 ;
                     6 ;
                     7 ;        DRIVER PROGRAM FOR ZIATECH'S ZT 85/38 IEEE 4
                     8 ;        INTERFACE BOARD FOR THE INTEL MULTIBUS USED
                     9 ;        IN CONJUNCTION WITH AN MULTIBUS COMPATIBLE
                    1Ø ;        CPU CAPABLE OF EXECUTING 8Ø8Ø TYPECODE.
                    11 ;        THIS PROGRAM IS VERSION 1 AND DOES NOT USE
                    12 ;        DMA OR INTERRUPTS WHILE DRIVING THE 9914 IN
                    13 ;        A CONTROLLER CONFIGURATION.
                    14 ;
                    15 ;        WRITTEN BY ALAN BEVERLY
                    16 ;        2/4/82    13ØØ HOURS
                    17 ;        ZIATECH CORPORATION
                    18 ;        SAN LUIS OBISPO, CA  934Ø1
                    19 ;
                    2Ø ;
                    21 ;**********************************************
                    22 ;
                    23 ;                        GENERAL CONTROL VALUES
                    24 ;
                    25 ;**********************************************
                    26 ;
                 =  27 $INCLUDE (:F1:EQU85.SRC)
                 =  28 ;
                 =  29 ;
                 =  3Ø ;                ZT 85/38 HARDWARE VALUES
                 =  31 ;
                 =  32 ;
                 =  33 ;        PORT SELECTIONS
  ØØ4Ø           =  34 BASE     EQU     4ØH      ; @ BASE PORT SELECTION
  ØØ4Ø           =  35 PRT17    EQU     BASE+ØØH; @ 9517 BASE PORT # (R/W)
  ØØ5Ø           =  36 PRT14    EQU     BASE+1ØH; @ 9914 BASE PORT # (R/W)
  ØØ58           =  37 INTR     EQU     BASE+18H; @ INTERRUPT ENABLE (W)
  ØØ58           =  38 INTRS    EQU     BASE+18H; @ INTERRUPT STATUS (R)
  ØØ5A           =  39 EOSR     EQU     BASE+1AH; @ EOS COMPARE (W)
  ØØ5A           =  4Ø MDAR     EQU     BASE+1AH; @ GPIB ADDRESS SWITCH (R)
  ØØ5C           =  41 SREST    EQU     BASE+1CH; @ SOFTWARE RESET (W)
  ØØ5E           =  42 LEDS     EQU     BASE+1EH; @ LED SELECT PORT (W)
                 =  43 ;
                 =  44 ;        INTERRUPT ENABLE EQUATES
  ØØØ1           =  45 EOSE     EQU     Ø1H      ; @ ENABLE  EOS COMPARE
  ØØ1Ø           =  46 EOSC     EQU     1ØH      ; @ DISABLE  EOS COMPARE
  ØØØ2           =  47 EOIM     EQU     Ø2H      ; @ ENABLE EOI INTERRUPT
  ØØ2Ø           =  48 EOIC     EQU     2ØH      ; @ DISABLE EOI INTERRUPT
  ØØ4Ø           =  49 EOSIC    EQU     4ØH      ; @ EOS+EOI F/F CLEAR
  ØØ8Ø           =  5Ø EOSIPC   EQU     8ØH      ; @ EOS+EOI+EOP F/F CLEAR
                 =  51 ;
                 =  52 ;        INTERRUPT MASK EQUATES
```

```
LOC  OBJ          SEQ         SOURCE STATEMENT

ØØ2Ø            =  53 I14M     EQU      2ØH       ; 9914 INTERRUPT MASK
ØØ4Ø            =  54 EOSIM    EQU      4ØH       ; EOS+EOI INTERRUPT MASK
ØØ8Ø            =  55 EOSIPM   EQU      8ØH       ; EOS+EOI+EOP F/F MASK
                =  56 ;
                =  57 ;        LED ENABLE EQUATES
ØØØ1            =  58 LED1E    EQU      Ø1H       ; ENABLE LED#1 FOR TALK
ØØ1Ø            =  59 LED1D    EQU      1ØH       ; DISABLE LED#1
ØØØ2            =  6Ø LED2E    EQU      Ø2H       ; ENABLE LED#2 FOR LISTEN
ØØ2Ø            =  61 LED2D    EQU      2ØH       ; DISABLE LED#2
                   62 ;
                =  63 $INCLUDE (:F1:EQU14.SRC)
                =  64 ;
                =  65 ;
                =  66 ;                 9914 CONTROL VALUES
                =  67 ;
                =  68 ;
                =  69 ;        REG #Ø INTERRUPT REG. Ø CONSTANTS (R/W)
ØØ5Ø            =  7Ø INTØ     EQU      PRT14+Ø ; INTERRUPT REG. Ø
ØØ5Ø            =  71 INTMØ    EQU      PRT14+Ø ; INTERRUPT REG Ø
ØØ8Ø            =  72 INTRØ    EQU      8ØH       ; REG Ø INTERRUPT MASK
ØØ4Ø            =  73 INTR1    EQU      4ØH       ; REG 1 INTERRUPT MASK
ØØ2Ø            =  74 BIM      EQU      2ØH       ; BYTE IN MASK
ØØ1Ø            =  75 BOM      EQU      1ØH       ; BYTE OUT MASK
ØØØ8            =  76 EOIMØ    EQU      Ø8H       ; EOI MASK
ØØØ4            =  77 SPASM    EQU      Ø4H       ; SERIAL POLL MASK
ØØØ2            =  78 RLCM     EQU      Ø2H       ; REMOTE/LOCAL CHANGE
                =  79 ;
                =  8Ø ;        REG #1 INTERRUPT REG. 1 CONSTANTS (R/W)
ØØ51            =  81 INT1     EQU      PRT14+1 ; INTERRUPT REG 1
ØØ51            =  82 INTM1    EQU      PRT14+1 ; INTERRUPT MASK REG. 1
ØØ8Ø            =  83 GETM     EQU      8ØH       ; GROUP EXECUTE TRIGGER
ØØ4Ø            =  84 ERRM     EQU      4ØH       ; INCOMPLETE HANDSHAKE
ØØ2Ø            =  85 UCGM     EQU      2ØH       ; UNIDENTIFIED COMMAND
ØØ1Ø            =  86 APTM     EQU      1ØH       ; ADDRESS PASS THROUGH
ØØØ8            =  87 DCASM    EQU      Ø8H       ; DEVICE CLEAR STATE
ØØØ4            =  88 MAM      EQU      Ø4H       ; (MLA OR MTA) AND NOT SPMS
ØØØ2            =  89 SRQM     EQU      Ø2H       ; SERVICE REQUEST
ØØØ1            =  9Ø IFCM     EQU      Ø1H       ; INTERFACE CLEAR MASK
                =  91 ;
                =  92 ;        REG #2 ADDRESSS STATUS (R)
ØØ52            =  93 ADRST    EQU      PRT14+2 ; ADDRESS STATUS REGISTER
ØØ8Ø            =  94 REMM     EQU      8ØH       ; REMOTE ENABLE MASK
ØØ4Ø            =  95 LLOM     EQU      4ØH       ; LOCAL LOCKOUT MASK
ØØ2Ø            =  96 ATNM     EQU      2ØH       ; ATTENTION STATUS
ØØ1Ø            =  97 LPASM    EQU      1ØH       ; PRI. LISTEN ADDRESSED
ØØØ8            =  98 TPASM    EQU      Ø8H       ; PRI. TALK ADDRESSED
ØØØ4            =  99 LADSM    EQU      Ø4H       ; LISTEN ADDRESSED
ØØØ2            = 1ØØ TADSM    EQU      Ø2H       ; TALK ADDRESSED
ØØØ1            = 1Ø1 ULPAM    EQU      Ø1H       ; LSM OF LAST ADD. REC.
                = 1Ø2 ;
                = 1Ø3 ;        REG #3 BUS STATUS (R)
ØØ53            = 1Ø4 BUSTR    EQU      PRT14+3 ; GPIB STATUS REGISTER
ØØ8Ø            = 1Ø5 ATNMB    EQU      8ØH       ; ATTENTION STATUS
ØØ4Ø            = 1Ø6 DAVM     EQU      4ØH       ; DATA VALID STATUS
ØØ2Ø            = 1Ø7 NDACM    EQU      2ØH       ; NO DATA ACCEPTED STATUS
```

```
 LOC  OBJ          SEQ         SOURCE STATEMENT

 0010              = 108 NRFDM   EQU      10H      ; NOT READY FOR DATA
 0008              = 109 EOIMK   EQU      08H      ; END OR IDENTIFY
 0004              = 110 SRQMM   EQU      04H      ; SEVICE REQUESTED
 0002              = 111 IFCMB   EQU      02H      ; INTERFACE CLEAR STATUS
 0001              = 112 RENM    EQU      01H      ; REMOTE ENABLED STATUS
                   = 113 ;
                   = 114 ;       REG #3 AUXILIARY COMMAND REGISTER (W)
 0053              = 115 AUXCMD  EQU      PRT14+3 ; AUX. CMD. REGISTER
 0080              = 116 SETM    EQU      80H      ; CLEAR/SET OPERATION (SET)
 007F              = 117 CLRM    EQU      7FH      ; CLEAR/SET OPERATION (CLEAR
                   = 118 ;
                   = 119 ;       SET/RESET COMMANDS
 0080              = 120 RESET   EQU      80H      ; CHIP RESET
 0000              = 121 RSTCLR  EQU      00H      ; STOP RESET
 0001              = 122 DACR    EQU      01H      ; RELEASE ACDS HOLDOFF
 0001              = 123 IVASR   EQU      01H      ; INVALID SECONDARY ADD
 0081              = 124 VSADR   EQU      81H      ; VALID SECONDARY ADD
 0083              = 125 HDFA    EQU      83H      ; HOLDOFF ON ALL DATA
 0003              = 126 HDACLR  EQU      03H      ; RELEASE HOLDOFF ON ALL
 0084              = 127 HDFE    EQU      84H      ; HOLDOFF ON EOI ONLY
 0004              = 128 HDECLR  EQU      04H      ; RELEASE HOLDOFF ON EOI
 0086              = 129 FGET    EQU      86H      ; FORCE GROUP EXECUTE TRIGGE
 0006              = 130 FGTCLR  EQU      06H      ; STOP GROUP EXECUTE TRIGGER
 0087              = 131 RTL     EQU      87H      ; RETURN TO LOCAL
 0007              = 132 RTLCLR  EQU      07H      ; DON'T RETURN TO LOCAL
 0089              = 133 LON     EQU      89H      ; LISTEN ONLY
 0009              = 134 LONCLR  EQU      09H      ; RESET LISTEN ONLY
 008A              = 135 TON     EQU      8AH      ; TALK ONLY
 000A              = 136 TONCLR  EQU      0AH      ; RESET TALK ONLY
 008E              = 137 RPP     EQU      8EH      ; REQ. PARALLEL POLL
 000E              = 138 RPPCLR  EQU      0EH      ; RESET PARALLEL POLL
 008F              = 139 SIC     EQU      8FH      ; SEND IFC
 000F              = 140 SICLR   EQU      0FH      ; RESET INTERFACE CLEAR
 0090              = 141 SRE     EQU      90H      ; SEND REM
 0010              = 142 SRECLR  EQU      10H      ; RESET REM
 0093              = 143 DAI     EQU      93H      ; DISABLE ALL INTERRUPTS
 0013              = 144 DAICLR  EQU      13H      ; ENABLE ALL INTERRUPTS
 0095              = 145 STDL    EQU      95H      ; SET T1 DELAY
 0015              = 146 STDCLR  EQU      15H      ; RESET T1 TIMER
 0096              = 147 SHDW    EQU      96H      ; SHADOW HANDSHAKE
 0016              = 148 SHDCLR  EQU      16H      ; RESET SHADOW HANDSHAKE
 0017              = 149 VSTDL   EQU      17H      ; SET FAST T1 DELAY
 0097              = 150 VSTCLR  EQU      97H      ; RESET FAST T1 DELAY
 0018              = 151 RSV2S   EQU      18H      ; SERVICE REQUEST #2
 0098              = 152 RSV2C   EQU      98H      ; CLEAR SR #2
                   = 153 ;
                   = 154 ;       PULSE TYPE COMMANDS
 0002              = 155 RHDF    EQU      02H      ; RELEASE RFD HOLDOFF
 0005              = 156 NBAF    EQU      05H      ; SUPPRESS BYTE SENT
 0008              = 157 FEOI    EQU      08H      ; SEND EOI WITH NEXT BYTE
 000B              = 158 GTS     EQU      0BH      ; GO TO STANDBY
 000C              = 159 TCA     EQU      0CH      ; TAKE CONTROL ASYNCH.
 000D              = 160 TCS     EQU      0DH      ; TAKE CONTROL SYNCH.
 0011              = 161 RQC     EQU      11H      ; REQUEST CONTROL
 0012              = 162 RLCT    EQU      12H      ; RELEASE CONTROL
```

```
LOC  OBJ         SEQ        SOURCE STATEMENT

0014             = 163 PTS      EQU     14H      ; PASS THROUGH NEXT SECONDAR
                 = 164 ;
                 = 165 ;        REG #4 ADDRESS REGISTER (W)
0054             = 166 ADDR     EQU     PRT14+4 ; 9914 ADDRESS REGISTER
0080             = 167 EDPA     EQU     80H      ; EN. DUAL PRI. ADD. MODE
0040             = 168 DAL      EQU     40H      ; DISABLE LISTEN MODE
0020             = 169 DAT      EQU     20H      ; DISABLE TALK MODE
0060             = 170 DALT     EQU     DAL+DAT ; DIS. BOTH TALK & LISTEN
                 = 171 ;
                 = 172 ;        REG #5 SERIAL POLL REGISTER (W)
0055             = 173 SPOLR    EQU     PRT14+5 ; SERIAL POLL REGISTER
0040             = 174 RSV      EQU     40H      ; REQUEST SERVICE
                 = 175 ;
                 = 176 ;        REG #6 COMMAND PASS THROUGH (R)
0056             = 177 CPTRG    EQU     PRT14+6 ; COMD. PASS THROUGH
                 = 178 ;
                 = 179 ;        REG #6 PARALLEL POLL REGISTER (W)
0056             = 180 PRPR     EQU     PRT14+6 ; PARALLEL POLL REG.
                 = 181 ;
                 = 182 ;        REG #7 DATA IN REGISTER (R)
0057             = 183 DIN      EQU     PRT14+7 ; DATA IN REGISTER
                 = 184 ;
                 = 185 ;        REG #7 DATA OUT REGISTER (W)
0057             = 186 DOUT     EQU     PRT14+7 ; DATA OUT REGISTER
                 = 187 ;
                 = 188 ;                GPIB COMMANDS
                 = 189 ;
0020             = 190 MLA      EQU     20H      ; MY LISTEN ADDRESS
0040             = 191 MTA      EQU     40H      ; MY TALK ADDRESS
003F             = 192 UNL      EQU     3FH      ; UNIVERSAL UNLISTEN
005F             = 193 UNT      EQU     5FH      ; UNIVERSAL UNTALK
0014             = 194 DCL      EQU     14H      ; DEVICE CLEAR
0008             = 195 GET      EQU     08H      ; GROUP EXECUTE TRIGGER
0011             = 196 LLO      EQU     11H      ; LOCAL LOCK OUT
0005             = 197 PPC      EQU     05H      ; PAR POLL CONFIGURE
0070             = 198 PPD      EQU     70H      ; PAR POLL DISABLE
0060             = 199 PPE      EQU     60H      ; PAR POLL ENABLE
0015             = 200 PPU      EQU     15H      ; PAR POLL UNCONFIGURE
0004             = 201 SDC      EQU     04H      ; SELECTED DEVICE CLEAR
0019             = 202 SPD      EQU     19H      ; SERIAL POLL DISABLE
0018             = 203 SPE      EQU     18H      ; SERIAL POLL ENABLE
0009             = 204 TCT      EQU     09H      ; TAKE CONTROL
                   205 ;
                   206 $EJECT
```

```
 LOC   OBJ          SEQ          SOURCE STATEMENT

                  = 207 $INCLUDE (:F1:MACRO.SRC)
                  = 208 ;
                  = 209 ;
                  = 210 ;*********************************************
                  = 211 ;
                  = 212 ;                          MACRO DEFINITIONS
                  = 213 ;
                  = 214 ;*********************************************
                  = 215 ;
                  = 216 ; NOTE: ALL MACROS DESTROY THE ACCUMULATOR.  ALL
                  = 217 ;       OTHER REGISTERS ARE PRESERVED.
                  = 218 ;
                  = 219 ;       SETF SETS FLAGS ON ACCUMULATOR
                  = 220 ;
                  = 221 SETF    MACRO
                  = 222         ORA     A
                  = 223         ENDM
                  = 224 ;
                  = 225 ;       WAITI WAITS FOR A BYTE INPUT
                  = 226 ;
                  = 227 WAITI   MACRO
                  = 228         LOCAL   WAITL
                  = 229 WAITL:  IN      INT0     ; GET INT0 STATUS
                  = 230         ANI     BIM      ; CHECK FOR BYTE IN
                  = 231         JZ      WAITL    ; WAIT UNTIL IT IS
                  = 232         ENDM
                  = 233 ;
                  = 234 ;       WAITO WAITS FOR BYTE OUT
                  = 235 ;
                  = 236 WAITO   MACRO
                  = 237         LOCAL   WAITL
                  = 238 WAITL:  IN      INT0     ; GET INT0 STATUS
                  = 239         ANI     BOM      ; CHECK FOR BYTE OUT
                  = 240         JZ      WAITL    ; WAIT UNTIL IT IS
                  = 241         ENDM
                  = 242 ;
                  = 243 ;       CMD SENDS A GIVEN MESSAGE TO THE AUXCMD REG
                  = 244 ;
                  = 245 CMD     MACRO   CMMD
                  = 246         MVI     A,CMMD  ; GET COMMAND
                  = 247         OUT     AUXCMD  ; SEND TO AUXCMD REGISTI
                  = 248         ENDM
                  = 249 ;
                  = 250 ;       RANGE JUMPS TO LABEL IF MEM. LOWER OR GREAT:
                  = 251 ;
                  = 252 RANGE   MACRO   LOWER,UPPER,LABEL
                  = 253         MOV     A,M     ; GET VALUE TO CHECK
                  = 254         CPI     LOWER   ; < OR = TO LOWER
                  = 255         JM      LABEL
                  = 256         CPI     UPPER+1 ; > OR = TO UPPER
                  = 257         JP      LABEL
                  = 258         ENDM
                  = 259 ;
                  = 260 ;       DELAY WILL WAIT A GIVEN AMOUNT OF TIME IN
                  = 261             ; MICROSECONDS ASSUMING A CLOCK OF 320N
```

```
LOC  OBJ          SEQ          SOURCE STATEMENT

                = 262 ;                 MAXIMUM DELAY IS 500MS.
                = 263 ;
                = 264 DELAY    MACRO    USEC
                = 265          LOCAL    WAITL
                = 266          PUSH     B          ; SAVE B
                = 267          LXI      B,USEC*3/23
                = 268 WAITL:   DCX      B          ; COUNT DOWN
                = 269          MOV      A,C        ; TEST FOR END
                = 270          ORA      B
                = 271          JNZ      WAITL
                = 272          POP      B          ; RESTORE B
                = 273          ENDM
                = 274 ;
                = 275 ;        ZERO WILL TEST A 16 BIT COUNTER FOR ZERO
                = 276 ;             THAT IS CONTAINED IN REGISTERS BC
                = 277 ;
                = 278 ZERO     MACRO    TDEST
                = 279          MOV      A,C        ; GET C REG
                = 280          ORA      B          ; TEST WITH B
                = 281          JZ       TDEST      ; ZERO SO JUMP
                = 282          ENDM
                = 283 ;
                = 284 ;        TALK CONTROLS LED #1 AS AN INDICATION
                = 285 ;             OF GPIB TALK ACTIVITY
                = 286 ;
                = 287 TALK     MACRO    LED1
                = 288          MVI      A,LED1     ; @ LOAD LED STATUS
                = 289          OUT      LEDS       ; @ SEND TO LED PORT
                = 290          ENDM
                = 291 ;
                = 292 ;        LISTEN CONTROLS LED #2 AS AN INDICATION
                = 293 ;               OF GPIB LISTEN ACTIVITY
                = 294 ;
                = 295 LISTEN   MACRO    LED2
                = 296          MVI      A,LED2     ; @ LOAD LED STATUS
                = 297          OUT      LEDS       ; @ SEND TO LED PORT
                = 298          ENDM
                = 299 ;
                = 300 ;        CLRA WILL ZERO OUT THE ACCUMULATOR
                = 301 ;
                = 302 CLRA     MACRO
                = 303          XRA      A          ; ZERO ACCUMULATOR
                = 304          ENDM
                  305 ;
                  306 $EJECT
```

```
LOC  OBJ          SEQ        SOURCE STATEMENT

                   307 ;*************************************************
                   308 ;
                   309 ;                      OPERATION OF SUBROUTINES
                   310 ;
                   311 ;**************************************************
                   312 ;
                   313 ;       ALL ROUTINES HAVE BEEN ORIGINED AT:
                   314 ;
5000               315         ORG     5000H   ; SYSTEM MEMORY
                   316 ;
                   317 ;       THE STACK POINTER WILL BE LOCATED AT:
                   318 ;
3E00               319 USER    EQU     3E00H   ; USER STACK AREA
                   320 ;
                   321 ;       WITH THE EOS CHARACTOR SAVED IN RAM AT:
                   322 ;
0001               323 EOS:    DS      01H     ; EOS CHARACTOR
                   324 ;
                   325 ;       ALL OF THE ROUTINES HAVE THESE COMMON AS-
                   326 ;       SUMPTIONS ABOUT THE STATE OF THE TMS 99   UI
                   327 ;       ROUTINE AND WILL EXIT THE ROUTINE IN AN
                   328 ;       IDENTICAL STATE WITHOUT REGARD TO THE ORDER
                   329 ;       IN WHICH CALLED.
                   330 ;
                   331 ;       9914:   BO IS OR HAS BEEN SET, ALL INTERRUP'
                   332 ;               ARE MASKED OFF, TON MODE, NOT LA,
                   333 ;               HOLDOFFS IN EFFECT OR ENABLED, NO
                   334 ;               HOLDOFFS WAITING FOR FINISH COMMAND
                   335 ;
                   336 ;       ALL STATUS INFORMATION IS OBTAINED BY POL-
                   337 ;       LING APPROPRIATE STATUS REGISTER.  NO INTER
                   338 ;       RUPTS AND NO DMA IS USED IN THIS VERSION.
                   339 ;
                   340 $EJECT
```

```
LOC  OBJ          SEQ          SOURCE STATEMENT

                = 341 $INCLUDE (:F1:INT14.SRC)
                = 342 ;
                = 343 ;
                = 344 ;***********************************************************
                = 345 ;
                = 346 ;                                INITIALIZE 9914
                = 347 ;
                = 348 ;***********************************************************
                = 349 ;
                = 350 ; INPUTS:         NONE
                = 351 ; OUTPUTS:        NONE
                = 352 ; CALLS:          NONE
                = 353 ; DESTROYS:       A,F/FS
                = 354 ;
                = 355 INIT:     CMD     RESET    ; RESET 9914
5001 3E80       = 356+          MVI     A,RESET ; GET COMMAND
5003 D353       = 357+          OUT     AUXCMD  ; SEND TO AUXCMD REGISTER
                = 358           CMD     RSTCLR
5005 3E00       = 359+          MVI     A,RSTCLR        ; GET COMMAND
5007 D353       = 360+          OUT     AUXCMD  ; SEND TO AUXCMD REGISTER
                = 361           CLRA
5009 AF         = 362+          XRA     A       ; ZERO ACCUMULATOR
500A D350       = 363           OUT     INTM0   ; DISABLE ALL MASK BITS
500C D351       = 364           OUT     INTM1
                = 365           CMD     SIC     ; SEND IFC & TAKE CONTROL
500E 3E8F       = 366+          MVI     A,SIC   ; GET COMMAND
5010 D353       = 367+          OUT     AUXCMD  ; SEND TO AUXCMD REGISTER
                = 368           DELAY   5000    ; FOR 5 MILLISECONDS
5012 C5         = 369+          PUSH    B       ; SAVE B
5013 018C02     = 370+          LXI     B,5000*3/23
5016 0B         = 371+??0001:   DCX     B       ; COUNT DOWN
5017 79         = 372+          MOV     A,C     ; TEST FOR END
5018 B0         = 373+          ORA     B
5019 C21650     = 374+          JNZ     ??0001
501C C1         = 375+          POP     B       ; RESTORE B
                = 376           CMD     SICLR   ; RESET IFC
501D 3E0F       = 377+          MVI     A,SICLR ; GET COMMAND
501F D353       = 378+          OUT     AUXCMD  ; SEND TO AUXCMD REGISTER
                = 379           CMD     VSTDL   ; SET FAST T1 MODE
5021 3E17       = 380+          MVI     A,VSTDL ; GET COMMAND
5023 D353       = 381+          OUT     AUXCMD  ; SEND TO AUXCMD REGISTER
                = 382           CMD     TON     ; TALK ONLY MODE
5025 3E8A       = 383+          MVI     A,TON   ; GET COMMAND
5027 D353       = 384+          OUT     AUXCMD  ; SEND TO AUXCMD REGISTER
                = 385           WAITO           ; WAIT UNTIL THERE
5029 DB50       = 386+??0002:   IN      INT0    ; GET INT0 STATUS
502B E610       = 387+          ANI     BOM     ; CHECK FOR BYTE OUT
502D CA2950     = 388+          JZ      ??0002  ; WAIT UNTIL IT IS
5030 C9         = 389           RET
                  390 ;
                  391 $EJECT
```

```
 LOC  OBJ          SEQ          SOURCE STATEMENT

                 = 392 $INCLUDE (:F1:SND14.SRC)
                 = 393 ;
                 = 394 ;
                 = 395 ;*************************************************
                 = 396 ;
                 = 397 ;                          SEND ROUTINE
                 = 398 ;
                 = 399 ;*************************************************
                 = 400 ;
                 = 401 ; INPUTS:        HL LISTENER LIST POINTER
                 = 402 ;                DE DATA BUFFER POINTER
                 = 403 ;                BC COUNT - 0 WILL CAUSE NO DATA SEN
                 = 404 ;                   EOS CHARACTER
                 = 405 ; OUTPUTS:       DATA AT BUFFER POINTER DE
                 = 406 ; CALLS:         NONE
                 = 407 ; DESTROYS:      A,BC,DE,HL,F/FS
                 = 408 ;
 5031 DB5A       = 409 SEND:     IN     MDAR     ; @ READ GPIB ADD DIP SW
 5033 E61F       = 410           ANI    1FH      ; @ GET GPIB ADDRESS
 5035 F640       = 411           ORI    MTA      ; MTA TO DISABLE ANY TALKERS
 5037 D357       = 412           OUT    DOUT     ; PREVIOUSLY SENDING
                 = 413           WAITO
 5039 DB50       = 414+??0003:   IN     INT0     ; GET INT0 STATUS
 503B E610       = 415+          ANI    BOM      ; CHECK FOR BYTE OUT
 503D CA3950     = 416+          JZ     ??0003   ; WAIT UNTIL IT IS
 5040 3E3F       = 417           MVI    A,UNL    ; SEND UNIVERSAL UNLISTEN
 5042 D357       = 418           OUT    DOUT     ; OUTPUT IT
                 = 419 SEND1:    RANGE  20H,3EH,SEND2    ; CHECK LISTENER LIS
 5044 7E         = 420+          MOV    A,M      ; GET VALUE TO CHECK
 5045 FE20       = 421+          CPI    20H      ; < OR = TO LOWER
 5047 FA5D50     = 422+          JM     SEND2
 504A FE3F       = 423+          CPI    3EH+1    ; > OR = TO UPPER
 504C F25D50     = 424+          JP     SEND2
                 = 425           WAITO           ; WAIT FOR PREVIOUS BO
 504F DB50       = 426+??0004:   IN     INT0     ; GET INT0 STATUS
 5051 E610       = 427+          ANI    BOM      ; CHECK FOR BYTE OUT
 5053 CA4F50     = 428+          JZ     ??0004   ; WAIT UNTIL IT IS
 5056 7E         = 429           MOV    A,M      ; GET THIS LISTENER
 5057 D357       = 430           OUT    DOUT     ; OUTPUT TO GPIB
 5059 23         = 431           INX    H        ; INCREMENT LIST POINTER
 505A C34450     = 432           JMP    SEND1    ; LOOP UNTILL DONE
                 = 433 SEND2:    CMD    GTS      ; GO TO STANDBY
 505D 3E0B       = 434+          MVI    A,GTS    ; GET COMMAND
 505F D353       = 435+          OUT    AUXCMD   ; SEND TO AUXCMD REGISTER
                 = 436           TALK   LED1E    ; @ ACTIVATE TALK LED
 5061 3E01       = 437+          MVI    A,LED1E  ; @ LOAD LED STATUS
 5063 D35E       = 438+          OUT    LEDS     ; @ SEND TO LED PORT
                 = 439           WAITO           ; FOR LAST LISTENER ADD.
 5065 DB50       = 440+??0005:   IN     INT0     ; GET INT0 STATUS
 5067 E610       = 441+          ANI    BOM      ; CHECK FOR BYTE OUT
 5069 CA6550     = 442+          JZ     ??0005   ; WAIT UNTIL IT IS
                 = 443           ZERO   SEND6    ; TEST FOR ZERO COUNT
 506C 79         = 444+          MOV    A,C      ; GET C REG .
 506D B0         = 445+          ORA    B        ; TEST WITH B
 506E CA9C50     = 446+          JZ     SEND6    ; ZERO SO JUMP
```

```
LOC  OBJ         SEQ          SOURCE STATEMENT

5071 3A0050     = 447          LDA     EOS      ; GET EOS
5074 6F         = 448          MOV     L,A      ; SAVE IT IN L
5075 0B         = 449 SEND3:   DCX     B        ; DECREMENT COUNT
5076 79         = 450          MOV     A,C      ; TEST FOR LAST BYTE
5077 B0         = 451          ORA     B        ; TEST
5078 CA8D50     = 452          JZ      SEND5    ; YES, SEND EOI
507B 1A         = 453          LDAX    D        ; GET DATA BYTE
507C BD         = 454          CMP     L        ; IS IT EOS ?
507D CA8D50     = 455          JZ      SEND5    ; IF CHAR = EOS, GO FINISH
5080 D357       = 456          OUT     DOUT     ; OUTPUT TO GPIB
5082 13         = 457          INX     D        ; INCR BUFFER POINTER
                = 458 SEND4:   WAITO
5083 DB50       = 459+??0006:  IN      INT0     ; GET INT0 STATUS
5085 E610       = 460+         ANI     BOM      ; CHECK FOR BYTE OUT
5087 CA8350     = 461+         JZ      ??0006   ; WAIT UNTIL IT IS
508A C37550     = 462          JMP     SEND3    ; STILL MORE DATA
                = 463 SEND5:   CMD     FEOI     ; SEND EOI WITH EOS
508D 3E08       = 464+         MVI     A,FEOI   ; GET COMMAND
508F D353       = 465+         OUT     AUXCMD   ; SEND TO AUXCMD REGISTER
5091 1A         = 466          LDAX    D        ; GET LAST BYTE
5092 D357       = 467          OUT     DOUT     ; OUTPUT TO GPIB
5094 13         = 468          INX     D        ; FOR CONSISTENCY
                = 469          WAITO
5095 DB50       = 470+??0007:  IN      INT0     ; GET INT0 STATUS
5097 E610       = 471+         ANI     BOM      ; CHECK FOR BYTE OUT
5099 CA9550     = 472+         JZ      ??0007   ; WAIT UNTIL IT IS
                = 473 SEND6:   CMD     TCA      ; TAKE CONTROL SYNCR.
509C 3E0C       = 474+         MVI     A,TCA    ; GET COMMAND
509E D353       = 475+         OUT     AUXCMD   ; SEND TO AUXCMD REGISTER
                = 476          TALK    LED1D    ; @ RESET TALK LED
50A0 3E10       = 477+         MVI     A,LED1D  ; @ LOAD LED STATUS
50A2 D35E       = 478+         OUT     LEDS     ; @ SEND TO LED PORT
                = 479          WAITO
50A4 DB50       = 480+??0008:  IN      INT0     ; GET INT0 STATUS
50A6 E610       = 481+         ANI     BOM      ; CHECK FOR BYTE OUT
50A8 CAA450     = 482+         JZ      ??0008   ; WAIT UNTIL IT IS
50AB C9         = 483          RET
                  484 ;
                  485 $EJECT
```

```
LOC  OBJ          SEQ         SOURCE STATEMENT

                  = 486 $INCLUDE (:F1:RCV14.SRC)
                  = 487 ;
                  = 488 ;
                  = 489 ;*****************************************************
                  = 490 ;
                  = 491 ;                            RECEIVE ROUTINE
                  = 492 ;
                  = 493 ;*****************************************************
                  = 494 ;
                  = 495 ; INPUTS:         HL TALKER BUFFER POINTER
                  = 496 ;                 DE DATA BUFFER POINTER
                  = 497 ;                 BC DATA COUNT (MAX BUFFER SIZE)
                  = 498 ;                    EOS CHARACTER
                  = 499 ; OUTPUTS:        FILLS BUFFER POINTED AT BY DE
                  = 500 ; CALLS:          NONE
                  = 501 ; DESTROYS:       A,BC,DE,HL,F/FS
                  = 502 ;
                  = 503 RECV:   RANGE   40H,5EH,RECV5
50AC 7E           = 504+          MOV     A,M       ; GET VALUE TO CHECK
50AD FE40         = 505+          CPI     40H       ; < OR = TO LOWER
50AF FA2E51       = 506+          JM      RECV5
50B2 FE5F         = 507+          CPI     5EH+1     ; > OR = TO UPPER
50B4 F22E51       = 508+          JP      RECV5
50B7 D357         = 509           OUT     DOUT      ; OUTPUT TALKER TO GPIB
                  = 510           WAITO             ; WAIT FOR BYTE OUT
50B9 DB50         = 511+??0009:   IN      INT0      ; GET INT0 STATUS
50BB E610         = 512+          ANI     BOM       ; CHECK FOR BYTE OUT
50BD CAB950       = 513+          JZ      ??0009    ; WAIT UNTIL IT IS
50C0 3E3F         = 514           MVI     A,UNL     ; STOP OTHER LISTENERS
50C2 D357         = 515           OUT     DOUT      ; OUTPUT TO GPIB
                  = 516           WAITO             ; WAIT FOR BYTE OUT
50C4 DB50         = 517+??0010:   IN      INT0      ; GET INT0 STATUS
50C6 E610         = 518+          ANI     BOM       ; CHECK FOR BYTE OUT
50C8 CAC450       = 519+          JZ      ??0010    ; WAIT UNTIL IT IS
50CB DB5A         = 520           IN      MDAR      ; @ READ GPIB ADDR DIP SW
50CD E61F         = 521           ANI     1FH       ; @ GET GPIB ADDRESS
50CF F620         = 522           ORI     MLA       ; SEND TO GPIB
50D1 D357         = 523           OUT     DOUT      ; OUTPUT TO GPIB
                  = 524           CMD     HDFA      ; HOLDOFF ON ALL DATA
50D3 3E83         = 525+          MVI     A,HDFA    ; GET COMMAND
50D5 D353         = 526+          OUT     AUXCMD    ; SEND TO AUXCMD REGISTEI
                  = 527           CMD     LON       ; LISTEN ONLY
50D7 3E89         = 528+          MVI     A,LON     ; GET COMMAND
50D9 D353         = 529+          OUT     AUXCMD    ; SEND TO AUXCMD REGISTER
                  = 530           LISTEN  LED2E     ; @ ENABLE LISTEN LED
50DB 3E02         = 531           MVI     A,LED2E   ; @ LOAD LED STATUS
50DD D35E         = 532+          OUT     LEDS      ; @ SEND TO LED PORT
                  = 533           WAITO             ; WAIT FOR MTA
50DF DB50         = 534+??0011:   IN      INT0      ; GET INT0 STATUS
50E1 E610         = 535+          ANI     BOM       ; CHECK FOR BYTE OUT
50E3 CADF50       = 536+          JZ      ??0011    ; WAIT UNTIL IT IS
                  = 537           CMD     GTS       ; GO TO STANDBY
50E6 3E0B         = 538+          MVI     A,GTS     ; GET COMMAND
50E8 D353         = 539+          OUT     AUXCMD    ; SEND TO AUXCMD REGISTER
50EA 210050       = 540           LXI     H,EOS     ; LOAD EOS ADDRESS
```

```
LOC  OBJ         SEQ          SOURCE STATEMENT

5ØED DB5Ø       = 541 RECV1:   IN      INTØ     ; GET INTERRUPT Ø STATUS
5ØEF E628       = 542          ANI     EOIMK+BIM       ; CHECK EOI OR BI
5ØF1 CAED5Ø     = 543          JZ      RECV1    ; WAIT UNTIL SET
5ØF4 E6Ø8       = 544          ANI     EOIMK    ; CHECK FOR EOI
5ØF6 C2ØE51     = 545          JNZ     RECV2    ; IF NOT EOI GET DATA
5ØF9 DB57       = 546          IN      DIN      ; GET DATA
5ØFB 12         = 547          STAX    D        ; STORE IN DATA BUFFER
5ØFC 13         = 548          INX     D        ; GET READY FOR NEXT BYTE
5ØFD BE         = 549          CMP     M        ; IS IT EOS
5ØFE CA1251     = 55Ø          JZ      RECV3    ; YES EOS, GO FINISH UP
51Ø1 ØB         = 551          DCX     B        ; DECREMENT COUNT
                = 552          ZERO    RECV4    ; CHECK FOR ZERO COUNT
51Ø2 79         = 553+         MOV     A,C      ; GET C REG
51Ø3 BØ         = 554+         ORA     B        ; TEST WITH B
51Ø4 CA1351     = 555+         JZ      RECV4    ; ZERO SO JUMP
                = 556          CMD     RHDF     ; RELEASE HOLDOFF
51Ø7 3EØ2       = 557+         MVI     A,RHDF   ; GET COMMAND
51Ø9 D353       = 558+         OUT     AUXCMD   ; SEND TO AUXCMD REGISTER
51ØB C3ED5Ø     = 559          JMP     RECV1    ; NOT ZERO, GET MORE DATA
51ØE DB57       = 56Ø RECV2:   IN      DIN      ; GET DATA BYTE
511Ø 12         = 561          STAX    D        ; STORE IN DATA BUFFER
5111 13         = 562          INX     D        ; GET READY FOR NEXT BYTE
5112 ØB         = 563 RECV3:   DCX     B        ; DECREMENT COUNT
                = 564 RECV4:   CMD     TCS      ; TAKE CONTROL SYNCHRONOUS
5113 3EØD       = 565+         MVI     A,TCS    ; GET COMMAND
5115 D353       = 566+         OUT     AUXCMD   ; SEND TO AUXCMD REGISTER
                = 567          LISTEN  LED2D    ; @ DISABLE LISTEN LED
5117 3E2Ø       = 568+         MVI     A,LED2D  ; @ LOAD LED STATUS
5119 D35E       = 569+         OUT     LEDS     ; @ SEND TO LED PORT
                = 57Ø          WAITO            ; WAIT FOR COMPLETION
511B DB5Ø       = 571+??ØØ12:  IN      INTØ     ; GET INTØ STATUS
511D E61Ø       = 572+         ANI     BOM      ; CHECK FOR BYTE OUT
511F CA1B51     = 573+         JZ      ??ØØ12   ; WAIT UNTIL IT IS
                = 574          CMD     RHDF     ; RELEASE HOLD OFF ON ALL
5122 3EØ2       = 575+         MVI     A,RHDF   ; GET COMMAND
5124 D353       = 576+         OUT     AUXCMD   ; SEND TO AUXCMD REGISTER
                = 577          CMD     TON      ; SET TO TALK ONLY
5126 3E8A       = 578+         MVI     A,TON    ; GET COMMAND
5128 D353       = 579+         OUT     AUXCMD   ; SEND TO AUXCMD REGISTER
                = 58Ø          CMD     HDACLR   ;
512A 3EØ3       = 581+         MVI     A,HDACLR        ; GET COMMAND
512C D353       = 582+         OUT     AUXCMD   ; SEND TO AUXCMD REGISTER
512E C9         = 583 RECV5:   RET
                  584 ;
                  585 $EJECT
```

```
LOC  OBJ         SEQ        SOURCE STATEMENT

               = 586 $INCLUDE (:F1:XFR14.SRC)
               = 587 ;
               = 588 ;
               = 589 ;**********************************************
               = 590 ;
               = 591 ;                        XFER ROUTINE
               = 592 ;
               = 593 ;**********************************************
               = 594 ;
               = 595 ; INPUTS:        HL DEVICE LIST POINTER
               = 596 ; OUTUTS:        NONE
               = 597 ; CALLS:         NONE
               = 598 ; DESTROYS:      A,HL,F/FS
               = 599 ;
               = 600 ; NOTE:          XFER WILL WORK ONLY IF THE TALKER
               = 601 ;                USES EOI TO TERMINATE THE TRANSFER.
               = 602 ;
               = 603 XFER:   RANGE    40H,5EH,XFER4
512F 7E        = 604+        MOV      A,M      ; GET VALUE TO CHECK
5130 FE40      = 605+        CPI      40H      ; < OR = TO LOWER
5132 FAA251    = 606+        JM       XFER4
5135 FE5F      = 607+        CPI      5EH+1    ; > OR = TO UPPER
5137 F2A251    = 608+        JP       XFER4
513A D357      = 609         OUT      DOUT     ; SEND TALKER TO GPIB
513C 23        = 610         INX      H        ; INCR LIST POINTER
               = 611         WAITO             ; WAIT FOR BYTE OUT
513D DB50      = 612+??0013: IN       INT0     ; GET INT0 STATUS
513F E610      = 613+        ANI      BOM      ; CHECK FOR BYTE OUT
5141 CA3D51    = 614+        JZ       ??0013   ; WAIT UNTIL IT IS
5144 3E3F      = 615         MVI      A,UNL    ; SEND UNIVERSAL UNLISTEN
5146 D357      = 616         OUT      DOUT     ; OUTPUT TO GPIB
               = 617 XFER1:  RANGE    20H,3EH,XFER2
5148 7E        = 618+        MOV      A,M      ; GET VALUE TO CHECK
5149 FE20      = 619+        CPI      20H      ; < OR = TO LOWER
514B FA6151    = 620+        JM       XFER2
514E FE3F      = 621+        CPI      3EH+1    ; > OR = TO UPPER
5150 F26151    = 622+        JP       XFER2
               = 623         WAITO    ; WAIT FOR BYTE OUT
5153 DB50      = 624+??0014: IN       INT0     ; GET INT0 STATUS
5155 E610      = 625+        ANI      BOM      ; CHECK FOR BYTE OUT
5157 CA5351    = 626+        JZ       ??0014   ; WAIT UNTIL IT IS
515A 7E        = 627         MOV      A,M      ; GET LISTENER
515B D357      = 628         OUT      DOUT     ; OUTPUT TO GPIB
515D 23        = 629         INX      H        ; INCR POINTER
515E C34851    = 630         JMP      XFER1    ; LOOP UNTIL NON-VALID LSNR
               = 631 XFER2:  CMD      SHDW     ; ACTIVATE SHADOW HANDSHAKE
5161 3E96      = 632+        MVI      A,SHDW   ; GET COMMAND
5163 D353      = 633+        OUT      AUXCMD   ; SEND TO AUXCMD REGISTER
               = 634         CMD      HDFE
5165 3E84      = 635+        MVI      A,HDFE   ; GET COMMAND
5167 D353      = 636+        OUT      AUXCMD   ; SEND TO AUXCMD REGISTER
               = 637         CMD      LON      ; SET TO LISTEN ONLY
5169 3E89      = 638+        MVI      A,LON    ; GET COMMAND
516B D353      = 639+        OUT      AUXCMD   ; SEND TO AUXCMD REGISTER
               = 640         LISTEN   LED2E    ; @ ENABLE LISTEN LED
```

```
LOC  OBJ          SEQ          SOURCE STATEMENT

516D 3E02        = 641+         MVI     A,LED2E ; @ LOAD LED STATUS
516F D35E        = 642+         OUT     LEDS    ; @ SEND TO LED PORT
                 = 643          WAITO           ; WAIT FOR COMPLETION
5171 DB50        = 644+??0015:  IN      INT0    ; GET INT0 STATUS
5173 E610        = 645+         ANI     BOM     ; CHECK FOR BYTE OUT
5175 CA7151      = 646+         JZ      ??0015  ; WAIT UNTIL IT IS
                 = 647          CMD     GTS     ; GO TO STANDBY
5178 3E0B        = 648+         MVI     A,GTS   ; GET COMMAND
517A D353        = 649+         OUT     AUXCMD  ; SEND TO AUXCMD REGISTER
517C DB50        = 650 XFER3:   IN      INT0    ; GET STATUS BYTE
517E E608        = 651          ANI     EOIMK   ; CHECK FOR EOI
5180 CA7C51      = 652          JZ      XFER3   ; WAIT FOR IT
                 = 653          CMD     TCS     ; TAKE CONTROL SYNCHRONOUSL
5183 3E0D        = 654+         MVI     A,TCS   ; GET COMMAND
5185 D353        = 655+         OUT     AUXCMD  ; SEND TO AUXCMD REGISTER
                 = 656          WAITO           ; WAIT FOR COMPLETION
5187 DB50        = 657+??0016:  IN      INT0    ; GET INT0 STATUS
5189 E610        = 658+         ANI     BOM     ; CHECK FOR BYTE OUT
518B CA8751      = 659+         JZ      ??0016  ; WAIT UNTIL IT IS
                 = 660          CMD     RHDF    ; RELEASE HOLDOFF
518E 3E02        = 661+         MVI     A,RHDF  ; GET COMMAND
5190 D353        = 662+         OUT     AUXCMD  ; SEND TO AUXCMD REGISTER
                 = 663          CMD     HDECLR
5192 3E04        = 664+         MVI     A,HDECLR        ; GET COMMAND
5194 D353        = 665+         OUT     AUXCMD  ; SEND TO AUXCMD REGISTER
                 = 666          CMD     SHDCLR  ; RESET SHADOW HANDSHAKE
5196 3E16        = 667+         MVI     A,SHDCLR        ; GET COMMAND
5198 D353        = 668+         OUT     AUXCMD  ; SEND TO AUXCMD REGISTER
                 = 669          LISTEN  LED2D   ; @ RESET LISTEN LED
519A 3E20        = 670+         MVI     A,LED2D ; @ LOAD LED STATUS
519C D35E        = 671+         OUT     LEDS    ; @ SEND TO LED PORT
                 = 672          CMD     TON     ; SET TO TALK ONLY
519E 3E8A        = 673+         MVI     A,TON   ; GET COMMAND
51A0 D353        = 674+         OUT     AUXCMD  ; SEND TO AUXCMD REGISTER
51A2 C9          = 675 XFER4:   RET
                   676 ;
                   677 $EJECT
```

```
LOC  OBJ          SEQ          SOURCE STATEMENT

                = 678 $INCLUDE (:F1:TRG14.SRC)
                = 679 ;
                = 680 ;
                = 681 ;**********************************************************
                = 682 ;
                = 683 ;                          TRIGGER ROUTINE
                = 684 ;
                = 685 ;**********************************************************
                = 686 ;
                = 687 ; INPUTS:        HL LISTENER LIST POINTER
                = 688 ; OUTPUTS:       NONE
                = 689 ; CALLS:         NONE
                = 690 ; DESTROYS:      A,HL F/FS
                = 691 ;
51A3 3E3F       = 692 TRIG:    MVI     A, UNL    ; GET UNIVERSAL UNLISTEN
51A5 D357       = 693          OUT     DOUT      ; OUTPUT TO GPIB
                = 694 TRIG1:   RANGE   20H,3EH,TRIG2
51A7 7E         = 695+         MOV     A,M       ; GET VALUE TO CHECK
51A8 FE20       = 696+         CPI     20H       ; < OR = TO LOWER
51AA FAC051     = 697+         JM      TRIG2
51AD FE3F       = 698+         CPI     3EH+1     ; > OR = TO UPPER
51AF F2C051     = 699+         JP      TRIG2
                = 700          WAITO             ; WAIT FOR BYTE OUTPUT
51B2 DB50       = 701+??0017:  IN      INT0      ; GET INT0 STATUS
51B4 E610       = 702+         ANI     BOM       ; CHECK FOR BYTE OUT
51B6 CAB251     = 703+         JZ      ??0017    ; WAIT UNTIL IT IS
51B9 7E         = 704          MOV     A,M       ; GET LISTENER
51BA D357       = 705          OUT     DOUT      ; SEND LISTNER TO GPIB
51BC 23         = 706          INX     H         ; INCREMENT POINTER
51BD C3A751     = 707          JMP     TRIG1     ; LOOP UNTIL NON-VALID LSNR
                = 708 TRIG2:   WAITO             ; WAIT FOR LAST BYTE OUT
51C0 DB50       = 709+??0018:  IN      INT0      ; GET INT0 STATUS
51C2 E610       = 710+         ANI     BOM       ; CHECK FOR BYTE OUT
51C4 CAC051     = 711+         JZ      ??0018    ; WAIT UNTIL IT IS
51C7 3E08       = 712          MVI     A,GET     ; GET GROUP EXECUTE TRIGGER
51C9 D357       = 713          OUT     DOUT      ; OUTPUT TO GPIB
                = 714          WAITO             ; WAIT FOR OUTPUT
51CB DB50       = 715+??0019:  IN      INT0      ; GET INT0 STATUS
51CD E610       = 716+         ANI     BOM       ; CHECK FOR BYTE OUT
51CF CACB51     = 717+         JZ      ??0019    ; WAIT UNTIL IT IS
51D2 C9         = 718          RET
                  719 ;
                  720 $EJECT
```

```
LOC  OBJ          SEQ         SOURCE STATEMENT

                = 721 $INCLUDE (:F1:DCR14.SRC)
                = 722 ;
                = 723 ;
                = 724 ;*****************************************************
                = 725 ;
                = 726 ;                          DEVICE CLEAR ROUTINE
                = 727 ;
                = 728 ;*****************************************************
                = 729 ;
                = 730 ; INPUTS:        HL LISTENER POINTER
                = 731 ; OUTPUTS:       NONE
                = 732 ; CALLS:         NONE
                = 733 ; DESTROYS:      A,HL,F/FS
                = 734 ;
51D3 3E3F       = 735 DCLR:   MVI     A,UNL     ; UNIVERSAL UNLISTEN
51D5 D357       = 736         OUT     DOUT      ; OUTPUT TO GPIB
                = 737 DCLR1:  RANGE   20H,3EH,DCLR2
51D7 7E         = 738+        MOV     A,M       ; GET VALUE TO CHECK
51D8 FE20       = 739+        CPI     20H       ; < OR = TO LOWER
51DA FAF051     = 740+        JM      DCLR2
51DD FE3F       = 741+        CPI     3EH+1     ; > OR = TO UPPER
51DF F2F051     = 742+        JP      DCLR2
                = 743         WAITO             ; WAIT FOR BYTE OUTPUT
51E2 DB50       = 744+??0020: IN      INT0      ; GET INT0 STATUS
51E4 E610       = 745+        ANI     BOM       ; CHECK FOR BYTE OUT
51E6 CAE251     = 746+        JZ      ??0020    ; WAIT UNTIL IT IS
51E9 7E         = 747         MOV     A,M       ; GET LISTENER
51EA D357       = 748         OUT     DOUT      ; SEND TO GPIB
51EC 23         = 749         INX     H         ; GET NEXT LISTENER
51ED C3D751     = 750         JMP     DCLR1     ; LOOP UNTIL A NON-VALID LSTN
                = 751 DCLR2:  WAITO             ; WAIT FOR LAST BYTE
51F0 DB50       = 752+??0021: IN      INT0      ; GET INT0 STATUS
51F2 E610       = 753+        ANI     BOM       ; CHECK FOR BYTE OUT
51F4 CAF051     = 754+        JZ      ??0021    ; WAIT UNTIL IT IS
51F7 3E04       = 755         MVI     A,SDC     ; SEND DEVICE CLEAR
51F9 D357       = 756         OUT     DOUT      ; TO ALL ADDRESSED LISTENERS
                = 757         WAITO             ; WAIT FOR BYTE OUTPUT
51FB DB50       = 758+??0022: IN      INT0      ; GET INT0 STATUS
51FD E610       = 759+        ANI     BOM       ; CHECK FOR BYTE OUT
51FF CAFB51     = 760+        JZ      ??0022    ; WAIT UNTIL IT IS
5202 C9         = 761         RET
                  762 ;
                  763 $EJECT
```

```
LOC  OBJ         SEQ          SOURCE STATEMENT

                 = 764 $INCLUDE (:F1:SRP14.SRC)
                 = 765 ;
                 = 766 ;
                 = 767 ;*************************************************
                 = 768 ;
                 = 769 ;                         SERIAL POLL ROUTINE
                 = 770 ;
                 = 771 ;*************************************************
                 = 772 ;
                 = 773 ; INPUTS:           HL TALKER LIST POINTER
                 = 774 ; OUTPUTS:          FILLS BUFFER POINTED TO BY DE
                 = 775 ; CALLS:            NONE
                 = 776 ; DESTROYS:         A,BC,DE,F/FS
                 = 777 ;
5203 3E3F        = 778 SPOL:     MVI      A,UNL      ; UNIVERSAL UNLISTEN
5205 D357        = 779           OUT      DOUT       ; OUTPUT TO GPIB
                 = 780           WAITO               ; WAIT FOR BYTE OUTPUT
5207 DB50        = 781+??0023:   IN       INT0       ; GET INT0 STATUS
5209 E610        = 782+          ANI      BOM        ; CHECK FOR BYTE OUT
520B CA0752      = 783+          JZ       ??0023     ; WAIT UNTIL IT IS
520E DB5A        = 784           IN       MDAR       ; @ GET DEVICE ADDRESS
5210 E61F        = 785           ANI      1FH        ; @ GET GPIB ADDRESS
5212 F620        = 786           ORI      MLA        ; SEND MLA
5214 D357        = 787           OUT      DOUT       ; OUTPUT TO GPIB
                 = 788           WAITO               ; WAIT FOR OUTPUT
5216 DB50        = 789+??0024:   IN       INT0       ; GET INT0 STATUS
5218 E610        = 790+          ANI      BOM        ; CHECK FOR BYTE OUT
521A CA1652      = 791+          JZ       ??0024     ; WAIT UNTIL IT IS
521D 3E18        = 792           MVI      A,SPE      ; SERIAL POLL ENABLE
521F D357        = 793           OUT      DOUT       ; OUTPUT TO GPIB
                 = 794           CMD      HDFA       ; HOLDOFF ON ALL DATA
5221 3E83        = 795+          MVI      A,HDFA     ; GET COMMAND
5223 D353        = 796+          OUT      AUXCMD     ; SEND TO AUXCMD REGISTER
                 = 797           WAITO               ; WAIT FOR BYTE OUTPUT
5225 DB50        = 798+??0025:   IN       INT0       ; GET INT0 STATUS
5227 E610        = 799+          ANI      BOM        ; CHECK FOR BYTE OUT
5229 CA2552      = 800+          JZ       ??0025     ; WAIT UNTIL IT IS
                 = 801 SPOL1:    RANGE    40H,5EH,SPOL2   ; CHECK TALKER LIST
522C 7E          = 802+          MOV      A,M        ; GET VALUE TO CHECK
522D FE40        = 803+          CPI      40H        ; < OR = TO LOWER
522F FA7352      = 804+          JM       SPOL2
5232 FE5F        = 805+          CPI      5EH+1      ; > OR = TO UPPER
5234 F27352      = 806+          JP       SPOL2
5237 7E          = 807           MOV      A,M        ; GET TALKER
5238 D357        = 808           OUT      DOUT       ; OUTPUT TO GPIB
523A 23          = 809           INX      H          ; INCREMENT TALKER POINTER
                 = 810           WAITO               ; WAIT FOR BYTE OUTPUT
523B DB50        = 811+??0026:   IN       INT0       ; GET INT0 STATUS
523D E610        = 812+          ANI      BOM        ; CHECK FOR BYTE OUT
523F CA3B52      = 813+          JZ       ??0026     ; WAIT UNTIL IT IS
                 = 814           CMD      LON        ; LISTEN ONLY
5242 3E89        = 815+          MVI      A,LON      ; GET COMMAND
5244 D353        = 816+          OUT      AUXCMD     ; SEND TO AUXCMD REGISTER
                 = 817           CMD      GTS        ; GO TO STANDBY
5246 3E0B        = 818+          MVI      A,GTS      ; GET COMMAND
```

```
LOC  OBJ         SEQ         SOURCE STATEMENT

5248 D353      = 819+         OUT     AUXCMD   ; SEND TO AUXCMD REGISTER
               = 82Ø          LISTEN  LED2E    ; @ ENABLE LISTEN LED
524A 3EØ2      = 821+         MVI     A,LED2E  ; @ LOAD LED STATUS
524C D35E      = 822+         OUT     LEDS     ; @ SEND TO LED PORT
               = 823          WAITI            ; WAIT FOR BYTE INPUT
524E DB5Ø      = 824+??ØØ27:  IN      INTØ     ; GET INTØ STATUS
525Ø E62Ø      = 825+         ANI     BIM      ; CHECK FOR BYTE IN
5252 CA4E52    = 826+         JZ      ??ØØ27   ; WAIT UNTIL IT IS
               = 827          CMD     TCS      ; TAKE CONTROL SYNC
5255 3EØD      = 828+         MVI     A,TCS    ; GET COMMAND
5257 D353      = 829+         OUT     AUXCMD   ; SEND TO AUXCMD REGISTER
               = 83Ø          LISTEN  LED2D    ; @ DISABLE LISTEN LED
5259 3E2Ø      = 831+         MVI     A,LED2D  ; @ LOAD LED STATUS
525B D35E      = 832+         OUT     LEDS     ; @ SEND TO LED PORT
               = 833          WAITO            ; WAIT FOR ACTION COMPLETE
525D DB5Ø      = 834+??ØØ28:  IN      INTØ     ; GET INTØ STATUS
525F E61Ø      = 835+         ANI     BOM      ; CHECK FOR BYTE OUT
5261 CA5D52    = 836+         JZ      ??ØØ28   ; WAIT UNTIL IT IS
5264 DB57      = 837          IN      DIN      ; GET SP RESPONSE BYTE
5266 12        = 838          STAX    D        ; STORE IT IN THE BUFFER
5267 13        = 839          INX     D        ; INCR BUFFER POINTER
               = 84Ø          CMD     TON      ; TALK ONLY MODE
5268 3E8A      = 841+         MVI     A,TON    ; GET COMMAND
526A D353      = 842+         OUT     AUXCMD   ; SEND TO AUXCMD REGISTER
               = 843          CMD     RHDF     ; RELEASE HOLD OFF
526C 3EØ2      = 844+         MVI     A,RHDF   ; GET COMMAND
526E D353      = 845+         OUT     AUXCMD   ; SEND TO AUXCMD REGISTER
527Ø C32C52    = 846          JMP     SPOL1    ; GET NEXT DEVICE ON LIST
5273 3E19      = 847 SPOL2:   MVI     A,SPD    ; SERIAL POLL DISABLE
5275 D357      = 848          OUT     DOUT     ; OUTPUT TO GPIB
               = 849          CMD     HDACLR   ;
5277 3EØ3      = 85Ø+         MVI     A,HDACLR        ; GET COMMAND
5279 D353      = 851+         OUT     AUXCMD   ; SEND TO AUXCMD REGISTER
               = 852          WAITO            ; WAIT FOR BYTE OUTPUT
527B DB5Ø      = 853+??ØØ29:  IN      INTØ     ; GET INTØ STATUS
527D E61Ø      = 854+         ANI     BOM      ; CHECK FOR BYTE OUT
527F CA7B52    = 855+         JZ      ??ØØ29   ; WAIT UNTIL IT IS
5282 C9        = 856          RET
                 857 ;
                 858 $EJECT
```

```
LOC  OBJ          SEQ          SOURCE STATEMENT

                 = 859 $INCLUDE (:F1:PPE14.SRC)
                 = 860 ;
                 = 861 ;
                 = 862 ;**********************************************
                 = 863 ;
                 = 864 ;                          PARALLEL POLL CONFIG. ROUTI
                 = 865 ;
                 = 866 ;**********************************************
                 = 867 ;
                 = 868 ; INPUTS:          HL LISTENER LIST POINTER
                 = 869 ;                  DE CONFIGURATION BYTE POINTER
                 = 870 ; OUTPUTS:         NONE
                 = 871 ; CALLS:           NONE
                 = 872 ; DESTROYS:        A,DE,HL,F/FS
                 = 873 ;
5283 3E3F        = 874 PPEN:    MVI     A,UNL     ; UNIVERSAL UNLISTEN
5285 D357        = 875          OUT     DOUT      ; OUTPUT TO GPIB
                 = 876 PPEN1:   RANGE 20H,3EH,PPEN2      ; CHECK LISTENER LI
5287 7E          = 877+         MOV     A,M       ; GET VALUE TO CHECK
5288 FE20        = 878+         CPI     20H       ; < OR = TO LOWER
528A FAB852      = 879+         JM      PPEN2
528D FE3F        = 880+         CPI     3EH+1     ; > OR = TO UPPER
528F F2B852      = 881+         JP      PPEN2
                 = 882          WAITO             ; WAIT FOR BYTE OUTPUT
5292 DB50        = 883+??0030:  IN      INT0      ; GET INT0 STATUS
5294 E610        = 884+         ANI     BOM       ; CHECK FOR BYTE OUT
5296 CA9252      = 885+         JZ      ??0030    ; WAIT UNTIL IT IS
5299 7E          = 886          MOV     A,M       ; GET LISTENER
529A D357        = 887          OUT     DOUT      ; OUTPUT TO GPIB
                 = 888          WAITO             ; WAIT FOR BYTE OUTPUT
529C DB50        = 889+??0031:  IN      INT0      ; GET INT0 STATUS
529E E610        = 890+         ANI     BOM       ; CHECK FOR BYTE OUT
52A0 CA9C52      = 891+         JZ      ??0031    ; WAIT UNTIL IT IS
52A3 3E05        = 892          MVI     A,PPC     ; PARALLEL POLL CONFIGURE
52A5 D357        = 893          OUT     DOUT      ; OUTPUT TO GPIB
                 = 894          WAITO             ; WAIT FOR BYTE OUTPUT
52A7 DB50        = 895+??0032:  IN      INT0      ; GET INT0 STATUS
52A9 E610        = 896+         ANI     BOM       ; CHECK FOR BYTE OUT
52AB CAA752      = 897+         JZ      ??0032    ; WAIT UNTIL IT IS
52AE 1A          = 898          LDAX    D         ; GET CONFIGURATION
52AF F660        = 899          ORI     PPE       ; OR WITH PPE
52B1 D357        = 900          OUT     DOUT      ; OUTPUT TO GPIB
52B3 23          = 901          INX     H         ; INCR BUFFER POINTERS
52B4 13          = 902          INX     D
52B5 C38752      = 903          JMP     PPEN1     ; LOOP UNTIL DONE
                 = 904 PPEN2:   WAITO             ; FOR LAST BYTE OUTPUT
52B8 DB50        = 905+??0033:  IN      INT0      ; GET INT0 STATUS
52BA E610        = 906+         ANI     BOM       ; CHECK FOR BYTE OUT
52BC CAB852      = 907+         JZ      ??0033    ; WAIT UNTIL IT IS
52BF C9          = 908          RET
                   909 ;
                   910 $EJECT
```

```
LOC  OBJ         SEQ         SOURCE STATEMENT

                = 911 $INCLUDE (:F1:PPD14.SRC)
                = 912 ;
                = 913 ;
                = 914 ;***************************************************
                = 915 ;
                = 916 ;                      PARALLEL POLL DISABLE ROUTINE
                = 917 ;
                = 918 ;***************************************************
                = 919 ;
                = 920 ; INPUTS:        HL LISTENER LIST POINTER
                = 921 ; OUTPUTS:       NONE
                = 922 ; CALLS:         NONE
                = 923 ; DESTROYS:      A,HL,F/FS
                = 924 ;
52C0 3E3F       = 925 PPDS:    MVI     A,UNL     ; UNIVERSAL UNLISTEN
52C2 D357       = 926          OUT     DOUT      ; OUTPUT TO GPIB
                = 927 PPDS1:   RANGE   20H,3EH,PPDS2    ; CHECK LSTNR. LIST
52C4 7E         = 928+         MOV     A,M       ; GET VALUE TO CHECK
52C5 FE20       = 929+         CPI     20H       ; < OR = TO LOWER
52C7 FADD52     = 930+         JM      PPDS2
52CA FE3F       = 931+         CPI     3EH+1     ; > OR = TO UPPER
52CC F2DD52     = 932+         JP      PPDS2
                = 933          WAITO             ; WAIT FOR BYTE OUTPUT
52CF DB50       = 934+??0034:  IN      INT0      ; GET INT0 STATUS
52D1 E610       = 935+         ANI     BOM       ; CHECK FOR BYTE OUT
52D3 CACF52     = 936+         JZ      ??0034    ; WAIT UNTIL IT IS
52D6 7E         = 937          MOV     A,M       ; GET LISTENER
52D7 D357       = 938          OUT     DOUT      ; OUTPUT TO GPIB
52D9 23         = 939          INX     H         ; INCR LIST POINTER
52DA C3C452     = 940          JMP     PPDS1     ; LOOP UNTIL INVALID LSNR
                = 941 PPDS2:   WAITO             ; FOR LAST BYTE OUTPUT
52DD DB50       = 942+??0035:  IN      INT0      ; GET INT0 STATUS
52DF E610       = 943+         ANI     BOM       ; CHECK FOR BYTE OUT
52E1 CADD52     = 944+         JZ      ??0035    ; WAIT UNTIL IT IS
52E4 3E70       = 945          MVI     A,PPD     ; PARALLEL POLL DISABLE
52E6 D357       = 946          OUT     DOUT      ; OUTPUT TO GPIB
                = 947          WAITO             ; WAIT FOR BYTE OUTPUT
52E8 DB50       = 948+??0036:  IN      INT0      ; GET INT0 STATUS
52EA E610       = 949+         ANI     BOM       ; CHECK FOR BYTE OUT
52EC CAE852     = 950+         JZ      ??0036    ; WAIT UNTIL IT IS
52EF C9         = 951          RET
                  952 ;
                  953 $EJECT
```

```
LOC  OBJ          SEQ          SOURCE STATEMENT

                 = 954 $INCLUDE (:F1:PPU14.SRC)
                 = 955 ;
                 = 956 ;
                 = 957 ;***************************************************
                 = 958 ;
                 = 959 ;                           PARALLEL POLL UNCONFIGURE
                 = 960 ;
                 = 961 ;***************************************************
                 = 962 ;
                 = 963 ; INPUTS:        NONE
                 = 964 ; OUTPTS:        NONE
                 = 965 ; CALLS:         NONE
                 = 966 ; DESTROYS:      A,F/FS
                 = 967 ;
52F0 3E15        = 968 PPUN:    MVI     A,PPU    ; PARALLEL POLL UNCONFIGURE
52F2 D357        = 969          OUT     DOUT     ; OUTPUT TO GPIB
                 = 970          WAITO            ; WAIT FOR BYTE OUTPUT
52F4 DB50        = 971+??0037:  IN      INT0     ; GET INT0 STATUS
52F6 E610        = 972+         ANI     BOM      ; CHECK FOR BYTE OUT
52F8 CAF452      = 973+         JZ      ??0037   ; WAIT UNTIL IT IS
52FB C9          = 974          RET
                   975 ;
                 = 976 $INCLUDE (:F1:PRP14.SRC)
                 = 977 ;
                 = 978 ;
                 = 979 ;**************************************************
                 = 980 ;
                 = 981 ;                           CONDUCT A PARALLEL POLL
                 = 982 ;
                 = 983 ;***************************************************
                 = 984 ;
                 = 985 ; INPUTS:        NONE
                 = 986 ; OUTPUTS:       A  PARALLEL POLL STATUS BYTE
                 = 987 ; CALLS:         NONE
                 = 988 ; DESTROYS:      A,F/FS
                 = 989 ;
                 = 990 PRPL:    CMD     RPP      ; EXECUTE PARALLEL POLL
52FC 3E8E        = 991+         MVI     A,RPP    ; GET COMMAND
52FE D353        = 992+         OUT     AUXCMD   ; SEND TO AUXCMD REGISTER
                 = 993          DELAY   125      ; WAIT FOR 125 MICRO-SECONDS
5300 C5          = 994+         PUSH    B        ; SAVE B
5301 011000      = 995+         LXI     B,125*3/23
5304 0B          = 996+??0038:  DCX     B        ; COUNT DOWN
5305 79          = 997+         MOV     A,C      ; TEST FOR END
5306 B0          = 998+         ORA     B
5307 C20453      = 999+         JNZ     ??0038
530A C1          =1000+         POP     B        ; RESTORE B
530B DB56        =1001          IN      CPTRG    ; GET PP STATUS
530D F5          =1002          PUSH    PSW      ; SAVE STATUS
                 =1003          CMD     RPPCLR   ; CLEAR PARALLEL POLL
530E 3E0E        =1004+         MVI     A,RPPCLR         ; GET COMMAND
5310 D353        =1005+         OUT     AUXCMD   ; SEND TO AUXCMD REGISTER
5312 F1          =1006          POP     PSW      ; RESTORE STATUS
5313 C9          =1007          RET
                  1008 ;
```

```
LOC  OBJ           SEQ          SOURCE STATEMENT

                  =1009 $INCLUDE (:F1:PCT14.SRC)
                  =1010 ;
                  =1011 ;
                  =1012 ;*********************************************************
                  =1013 ;
                  =1014 ;                              PASS CONTROL ROUTINE
                  =1015 ;
                  =1016 ;*********************************************************
                  =1017 ;
                  =1018 ;  INPUTS:        HL POINTER TO TALKER LIST
                  =1019 ;  OUTPUTS:       NONE
                  =1020 ;  CALLS:         WAITO
                  =1021 ;  DESTROYS:      A,B,HL,F/FS
                  =1022 ;  DESCRIPTION:   THIS ROUTINE WILL PASS CONTROL FROM
                  =1023 ;                 THE ACTIVE CONTROLLER (IE US) TO THE
                  =1024 ;                 DEVICE SPECIFIED IN THE TALKER LIST.
                  =1025 ;                 THE TALKER ADDRESS IS CHECKED FOR A
                  =1026 ;                 VALID TALKER ADDRESS AND THAT IT IS
                  =1027 ;                 NOT OUR ADDRESS (IF EITHER TEST FAIL
                  =1028 ;                 NO ACTION IS TAKEN).  THE TAKE CON-
                  =1029 ;                 TROL COMMAND (TCT) IS SENT AND THE
                  =1030 ;                 MASK FOR UNIDENTIFIED CMD INTERRUPT
                  =1031 ;                 IS ENABLED (NEEDED TO RECEIVE CONTRO
                  =1032 ;                 IF PASSED TO US).  WE WILL GO IDLE
                  =1033 ;                 WHEN THE TCT MESSAGE IS HANDSHAKEN
                  =1034 ;                 BY THE DEVICE RECEIVING CONTROL.
                  =1035 ;
5314 DB5A         =1036 PCTL:    IN      MDAR     ; GET MY GPIB ADDRESS
5316 E61F         =1037          ANI     1FH      ; GET GPIB ADDRESS
5318 47           =1038          MOV     B,A      ; SAVE GPIB ADDRESS
5319 F640         =1039          ORI     MTA      ; GET MY TALK ADDRESSS
                  =1040          RANGE   40H,5EH,PCTL1    ; CHECK TALKER LIST
531B 7E           =1041+         MOV     A,M      ; GET VALUE TO CHECK
531C FE40         =1042+         CPI     40H      ; < OR = TO LOWER
531E FA4E53       =1043+         JM      PCTL1
5321 FE5F         =1044+         CPI     5EH+1    ; > OR = TO UPPER
5323 F24E53       =1045+         JP      PCTL1
5326 B8           =1046          CMP     B        ; IS IT MY TALKER ADDRESS
5327 CA4E53       =1047          JZ      PCTL1    ; YES, JUST RETURN
532A D357         =1048          OUT     DOUT     ; NO, OUTPUT TO GPIB
                  =1049          WAITO            ; WAIT FOR OUTPUT
532C DB50         =1050+??0039:  IN      INT0     ; GET INT0 STATUS
532E E610         =1051+         ANI     BOM      ; CHECK FOR BYTE OUT
5330 CA2C53       =1052+         JZ      ??0039   ; WAIT UNTIL IT IS
5333 3E09         =1053          MVI     A,TCT    ; TAKE CONTROL MESSAGE
5335 D357         =1054          OUT     DOUT     ; OUTPUT TO GPIB
                  =1055          WAITO            ; WAIT FOR BYTE OUTPUT
5337 DB50         =1056+??0040:  IN      INT0     ; GET INT0 STATUS
5339 E610         =1057+         ANI     BOM      ; CHECK FOR BYTE OUT
533B CA3753       =1058+         JZ      ??0040   ; WAIT UNTIL IT IS
                  =1059          CMD     TONCLR   ; RELEASE TALK ONLY
533E 3E0A         =1060+         MVI     A,TONCLR         ; GET COMMAND
5340 D353         =1061+         OUT     AUXCMD   ; SEND TO AUXCMD REGISTER
5342 3E20         =1062          MVI     A,UCGM   ; EN. UNIDENTIFIED CMD INTR.
5344 D351         =1063          OUT     INTM1    ;
```

```
LOC  OBJ          SEQ         SOURCE STATEMENT

5346 78          =1Ø64         MOV     A,B       ; GET GPIB ADDRESS
5347 D354        =1Ø65         OUT     ADDR      ; ENABLE TALK AND LISTEN
                 =1Ø66         CMD     RLCT      ; GO TO IDLE
5349 3E12        =1Ø67+        MVI     A,RLCT    ; GET COMMAND
534B D353        =1Ø68+        OUT     AUXCMD    ; SEND TO AUXCMD REGISTER
534D 23          =1Ø69         INX     H         ; FOR CONSISTENCY
534E C9          =1Ø7Ø PCTL1:  RET
                  1Ø71 ;
                  1Ø72 $EJECT
```

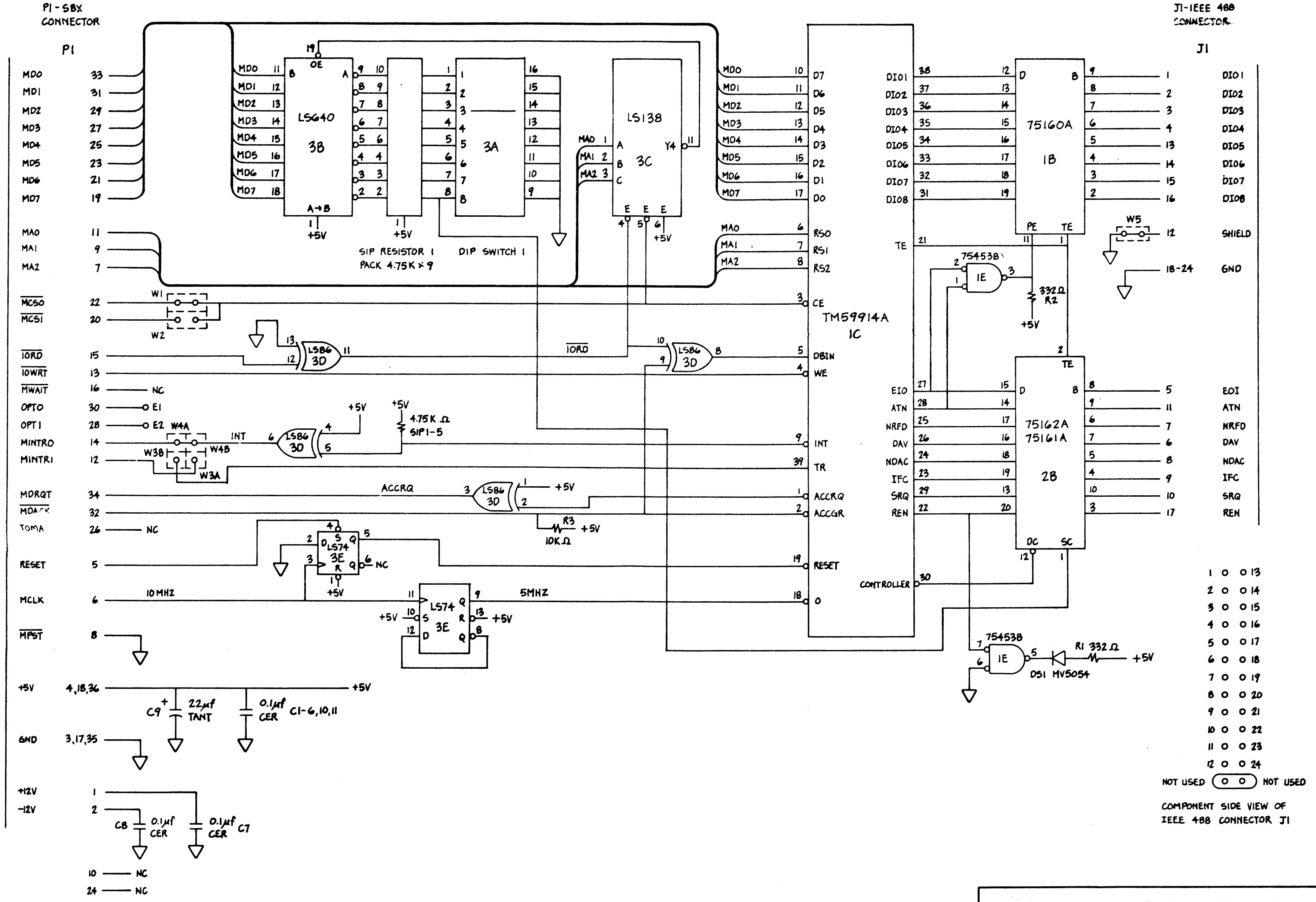
```
LOC  OBJ         SEQ         SOURCE STATEMENT

                 =1073 $INCLUDE (:F1:RCT14.SRC)
                 =1074 ;
                 =1075 ;
                 =1076 ;************************************************
                 =1077 ;
                 =1078 ;                          RECEIVE CONTROL
                 =1079 ;
                 =1080 ;************************************************
                 =1081 ;
                 =1082 ; INPUTS:        NONE
                 =1083 ; OUTPUTS:       A = 0 DO NOT TAKE CONTROL
                 =1084 ;                A <> 0 TAKE CONTROL
                 =1085 ; CALLS:         WAITO
                 =1086 ; DESTROYS:      A,F/FS
                 =1087 ; DESCRIPTION:   THIS ROUTINE WILL RECEIVE CONTROL
                 =1088 ;                FROM THE CURRENTLY ACTIVE CONTROL-
                 =1089 ;                LER.  THE UNIDENTIFIED COMMAND INTER
                 =1090 ;                RUPT (UCGM) MUST HAVE BEEN ENABLED
                 =1091 ;                EITHER DURING INITIALIZATION OR WHEN
                 =1092 ;                CONTROL WAS PASSED.  THUS THIS ROU-
                 =1093 ;                TINE IS INVOKED WHEN A UCGN INTER-
                 =1094 ;                RUPT IS HANDLED BY THE CPU.  IF
                 =1095 ;                THE UNIDENTIFIED COMMAND WAS NOT A
                 =1096 ;                TAKE CONTROL COMMAND (TCT) OR IF IT
                 =1097 ;                WAS AND WE WERE NOT TALK ADDRESSED
                 =1098 ;                THEN THIS ROUTINE OUTPUTS A = 0.
                 =1099 ;                IF IT IS A TCT COMMAND AND WE ARE
                 =1100 ;                TALK ADDRESSED THEN ALL INTERRUPT
                 =1101 ;                MASKS ARE CLEARED AND WE WILL RE-
                 =1102 ;                QUEST CONTROL OF THE BUS AND RETURN
                 =1103 ;                A < > 0 WHEN CONTROL IS TAKEN.
                 =1104 ;                NORMALLY SOME ADVANCE WARNING OF
                 =1105 ;                IMPENDING PASS CONTROL SHOULD BE
                 =1106 ;                GIVEN TO US BY THE CONTROLLER WITH
                 =1107 ;                OTHER USEFUL INFO.  THIS PROTOCOL
                 =1108 ;                IS SITUATION SPECIFIC AND IS NOT
                 =1109 ;                COVERED HERE.
                 =1110 ;
                 =1111 RCTL:
534F DB51        =1112         IN      INT1     ; GET STATUS
5351 E620        =1113         ANI     UCGM     ; UNIDENTIFIED COMMAND?
5353 CA8453      =1114         JZ      RCTL2    ; YES, GO RETURN
5356 DB56        =1115         IN      CPTRG    ; YES, GET COMMAND PASS THRU
5358 FE09        =1116         CPI     TCT      ; IS IT TAKE CONTROL?
535A C27F53      =1117         JNZ     RCTL1    ; NO, GO RETURN INVALID
535D DB52        =1118         IN      ADRST    ; GET ADDRESS STATUS
535F E602        =1119         ANI     TADSM    ; ARE WE TALK ADDRESSED?
5361 CA7F53      =1120         JZ      RCTL1    ; NO, GO RETURN INVALID
                 =1121         CLRA
5364 AF          =1122+        XRA     A        ; ZERO ACCUMULATOR
5365 D350        =1123         OUT     INTM0    ; CLEAR INTERRUPT MASKS
5367 D351        =1124         OUT     INTM1
                 =1125         CMD     RQC      ; REQUEST CONTROL
5369 3E11        =1126+        MVI     A,RQC    ; GET COMMAND
536B D353        =1127+        OUT     AUXCMD   ; SEND TO AUXCMD REGISTER
```

```
LOC  OBJ         SEQ          SOURCE STATEMENT

                =1128          WAITO                ; WAIT FOR CONTROL
536D DB5Ø       =1129+??ØØ41:  IN       INTØ        ; GET INTØ STATUS
536F E61Ø       =113Ø+         ANI      BOM         ; CHECK FOR BYTE OUT
5371 CA6D53     =1131+         JZ       ??ØØ41      ; WAIT UNTIL IT IS
                =1132          CMD      TON         ; SET TALK ONLY
5374 3E8A       =1133+         MVI      A,TON       ; GET COMMAND
5376 D353       =1134+         OUT      AUXCMD      ; SEND TO AUXCMD REGISTER
                =1135          CMD      DACR        ; ACKNOWLEDGE CMD PASS THRU
5378 3EØ1       =1136+         MVI      A,DACR      ; GET COMMAND
537A D353       =1137+         OUT      AUXCMD      ; SEND TO AUXCMD REGISTER
537C C38453     =1138          JMP      RCTL2
                =1139 RCTL1:   CMD      DACR        ; ACKNOWLEDGE CMD PASS THRU
537F 3EØ1       =114Ø+         MVI      A,DACR      ; GET COMMAND
5381 D353       =1141+         OUT      AUXCMD      ; SEND TO AUXCMD REGISTER
                =1142          CLRA                 ; INVALID RETURN CODE
5383 AF         =1143+         XRA      A           ; ZERO ACCUMULATOR
5384 C9         =1144 RCTL2:   RET
                 1145 ;
                 1146 $EJECT
```

```
LOC  OBJ          SEQ          SOURCE STATEMENT

                 =1147 $INCLUDE (:F1:SCM14.SRC)
                 =1148 ;
                 =1149 ;
                 =1150 ;**********************************************
                 =1151 ;
                 =1152 ;                       SEND COMMAND STRING
                 =1153 ;
                 =1154 ;**********************************************
                 =1155 ;
                 =1156 ; INPUTS:        DE  COMMAND STRING POINTER
                 =1157 ; OUTPUTS:       NONE
                 =1158 ; CALLS:         WAITO
                 =1159 ; DESTROYS:      A,DE,F/FS
                 =1160 ;
5385 1A          =1161 SCND:   LDAX    D         ; GET COMMAND
5386 FEFF        =1162         CPI     0FFH      ; IS IT LIST END?
5388 CA9853      =1163         JZ      SCMD1     ; YES, GO RETURN
538B D357        =1164         OUT     DOUT      ; NO, SEND TO GPIB
538D 13          =1165         INX     D         ; INCR POINTER
                 =1166         WAITO             ; WAIT TILL DONE
538E DB50        =1167+??0042: IN      INT0      ; GET INT0 STATUS
5390 E610        =1168+        ANI     BOM       ; CHECK FOR BYTE OUT
5392 CA8E53      =1169+        JZ      ??0042    ; WAIT UNTIL IT IS
5395 C38553      =1170         JMP     SCND      ; LOOP TILL DONE
5398 13          =1171 SCMD1:  INX     D         ; FOR CONSISTENCY
5399 C9          =1172         RET
                  1173 ;
                 =1174 $INCLUDE (:F1:SQD14.SRC)
                 =1175 ;
                 =1176 ;
                 =1177 ;**********************************************
                 =1178 ;
                 =1179 ;                       SRQ OCCURRED ROUTINE
                 =1180 ;
                 =1181 ;**********************************************
                 =1182 ;
                 =1183 ; INPUTS:        NONE
                 =1184 ; OUTPUTS:       A = 0 NO SRQ OCCURRED
                 =1185 ;                A < > 0 SRQ OCCURRED
                 =1186 ; CALLS:         NONE
                 =1187 ; DESTROYS:      A,F/FS
                 =1188 ;
539A DB51        =1189 SRQD:   IN      INT1      ; GET INTR1 STATUS
539C E602        =1190         ANI     SRQM      ; CHECK FOR SRQ
539E C2A253      =1191         JNZ     SRQD1     ; SRQ, GO RETURN
                 =1192         CLRA              ; NO SRQ, SO CLEAR ACC.
53A1 AF          =1193+        XRA     A         ; ZERO ACCUMULATOR
53A2 C9          =1194 SRQD1:  RET
                  1195 ;
                  1196 $EJECT
```

```
LOC  OBJ          SEQ          SOURCE STATEMENT

                 =1197 $INCLUDE (:F1:REM14.SRC)
                 =1198 ;
                 =1199 ;
                 =12ØØ ;**************************************************
                 =12Ø1 ;
                 =12Ø2 ;                              REMOTE ENABLE ROUTINE
                 =12Ø3 ;
                 =12Ø4 ;**************************************************
                 =12Ø5 ;
                 =12Ø6 ; INPUTS:          NONE
                 =12Ø7 ; OUTPUTS:         NONE
                 =12Ø8 ; CALLS:           NONE
                 =12Ø9 ; DESTROYS:        A,F/FS
                 =121Ø ;
                 =1211 REME:    CMD       SRE      ; ASSERT REMOTE ENABLE
53A3 3E9Ø        =1212+         MVI       A,SRE    ; GET COMMAND
53A5 D353        =1213+         OUT       AUXCMD   ; SEND TO AUXCMD REGISTER
53A7 C9          =1214          RET
                  1215 ;
                 =1216 $INCLUDE (:F1:LOC14.SRC)
                 =1217 ;
                 =1218 ;
                 =1219 ;**************************************************
                 =122Ø ;
                 =1221 ;                              LOCAL ROUTINE
                 =1222 ;
                 =1223 ;**************************************************
                 =1224 ;
                 =1225 ; INPUTS:          NONE
                 =1226 ; OUTPUTS:         NONE
                 =1227 ; CALLS:           NONE
                 =1228 ; DESTROYS:        A,F/FS
                 =1229 ;
                 =123Ø LOCL:    CMD       SRECLR   ; RESET REM LINE
53A8 3E1Ø        =1231+         MVI       A,SRECLR        ; GET COMMAND
53AA D353        =1232+         OUT       AUXCMD   ; SEND TO AUXCMD REGISTER
53AC C9          =1233          RET
                  1234 ;
                  1235 $EJECT
```

```
LOC  OBJ          SEQ        SOURCE STATEMENT

                 =1236 $INCLUDE (:F1:IFC14.SRC)
                 =1237 ;
                 =1238 ;
                 =1239 ;*************************************************
                 =1240 ;
                 =1241 ;                         INTERFACE CLEAR ROUTINE
                 =1242 ;
                 =1243 ;*************************************************
                 =1244 ;
                 =1245 ; INPUTS:        NONE
                 =1246 ; OUTUTS:        NONE
                 =1247 ; CALLS:         NONE
                 =1248 ; DESTROYS:      A,F/FS
                 =1249 ;
                 =1250 IFCL:    CMD     SIC      ; SEND INTERFACE CLEAR
53AD 3E8F        =1251+         MVI     A,SIC    ; GET COMMAND
53AF D353        =1252+         OUT     AUXCMD   ; SEND TO AUXCMD REGISTER
                 =1253          DELAY   5000     ; WAIT 5 MS
53B1 C5          =1254+         PUSH    B        ; SAVE B
53B2 018C02      =1255+         LXI     B,5000*3/23
53B5 0B          =1256+??0043: DCX      B        ; COUNT DOWN
53B6 79          =1257+         MOV     A,C      ; TEST FOR END
53B7 B0          =1258+         ORA     B
53B8 C2B553      =1259+         JNZ     ??0043
53BB C1          =1260+         POP     B        ; RESTORE B
                 =1261          CMD     SICLR    ; RESET IFC
53BC 3E0F        =1262+         MVI     A,SICLR  ; GET COMMAND
53BE D353        =1263+         OUT     AUXCMD   ; SEND TO AUXCMD REGISTER
53C0 C9          =1264          RET
                  1265 ;
                 =1266 $INCLUDE (:F1:ZT85R.SRC)
                 =1267 ;
                 =1268 ;
                 =1269 ;*************************************************
                 =1270 ;
                 =1271 ;                         ZT85 HARDWARE RESET
                 =1272 ;
                 =1273 ;*************************************************
                 =1274 ;
                 =1275 ; INPUTS:        NONE
                 =1276 ; OUTPUTS:       NONE
                 =1277 ; CALLS:         NONE
                 =1278 ; DESTROYS:      NONE
                 =1279 ;
53C1 D35C        =1280 ZT85R:   OUT     SREST    ; @ RESET ZT85
53C3 C9          =1281          RET
                  1282 ;
                  1283 ;
                  1284          END
```



Ziatech Corporation

ZIATECH CORPORATION

REVISIONS	TITLE ZSBX 20 IEEE 488 MULTIMODULE REV B	
10 SEPT. 82	SCALE NONE	DRAWING NO
R.E. SHEELY		