Twenty Dollars

Cromemco

GPIB

General Purpose Interface Board

Instruction Manual

Cromemco®

GPIB

General Purpose Interface Board

Instruction Manual

CROMEMCO, Inc.
280 Bernardo Avenue
Mountain View, CA. 94043

Part No. 023-2007

April 1983

This manual was produced using a Cromemco System Three computer with a Cromemco HDD-22 Hard Disk Storage System running under the Cromemco Cromix® Operating System. The text was edited with the Cromemco Cromix Screen Editor. The edited text was proofread by the Cromemco SpellMaster™ Program and formatted by the Cromemco Word Processing System Formatter II. Camera-ready copy was printed on a Cromemco 3355B printer.

TABLE OF CONTENTS

LIST OF FIGURES AND TABLES

TECHNICAL SPECIFICATIONS	
Applications:	Intelligent IEEE 488 bus to Host S-100 bus interface, or intelligent IEEE 488 stand alone Controller
IEEE-488 Functions:	**C** -- Controller with Pass Control capability; **T, TE, L, LE** -- Talker and/or Listener; **SH, AH** -- automatic Source and Acceptor Handshake; **DC, DT** -- Device Clear and Device Trigger; **SR** -- Service Request; **PP** -- Parallel Poll; **RL** -- Remote/Local with Remote Lockout
Processor:	Z-80A clocked at 4.000 MHz
ROM Memory:	Socket space for 4 Kbytes of TI 2516, Intel 2716, or their generic equivalents (user supplied)
RAM Memory:	4 Kbytes of 9124 (1K x 4) static RAM (included)
S-100 Bus Interface:	Four 8-bit parallel I/O ports; the GPIB board may issue non-maskable and/or vectored maskable interrupt requests to the Host; the Host may issue maskable interrupt requests to the GPIB board
GPIB Bus Interface:	The 24 IEEE 488 bus lines interface to GPIB board connector J2
Utility Interface:	One 8-bit parallel input port, and one 8-bit parallel output port provide a general purpose interface at GPIB board connector J1
LSI Device Types:	Z-80A CPU, TMS 9914 GPIB Adapter
Power Requirements:	+8 VDC @ 1.5 Amps (max)
Operating Environment:	0 - 55 degrees Celsius

ABOUT THIS MANUAL

This manual provides operating instructions for Cromemco's intelligent GPIB Interface board. Two LSI devices supply most of the interface's capabilities: a Z-80A central processing unit, and a TMS 9914 GPIB Adapter. It's assumed that the reader has technical documentation for these parts (see Mostek publication MK 3880 Central Processing Unit, 1977, and Texas Instruments publication TMS 9914 GPIB Adapter Preliminary Data Manual, 1979), and also that the reader is familiar with **Z-80 Assembly Language** and the **IEEE Std 488-1978.**

The manual chapters discuss all board functions except internal functions of these two LSI devices. Included are topics such as the board's Reset State, onboard ROM and RAM memory, the I/O mapping of LSI internal registers, and the board's interrupt structure.

After reading this manual and the reference documentation, the GPIB Interface user should be able to:

1. Design and write GPIB interface and host Z-80 software.

2. Configure the GPIB board. This means:

   a. Select the board's GPIB Talk and Listen address with switch SW1;

   b. Select the board's S-100 base I/O address with switch SW2;

   c. Optionally connect the board to Cromemco's S-100 interrupt priority daisy chain using connector J3;

   d. Install the 2516 firmware from step 1 in board sockets ROM0 and ROM1;

   e. Interface the board to the IEEE 488 instrumentation bus using connector J2.

3. Test and debug the host software and GPIB resident firmware.

Two conventions will be used consistently throughout the remainder of this manual. First, positive logic is assumed. **Reset** means logic 0, and **set** means logic 1 when these terms apply to bit states. Secondly, the acronym **GPIBB** will denote the **GPIB Board** itself, and its functions, as opposed to GPIB interface bus functions.

Chapter 1

INTRODUCTION

The Cromemco General Purpose Interface Bus (GPIB) Board is an intelligent interface between a host system S-100 bus and an IEEE 488 instrumentation bus (see Figure 1-1). Two TI 2516 EPROM sockets (4 Kbytes) supply the GPIBB program store, and 4 Kbytes of RAM memory are supplied for data buffering, scratchpad memory and the Z-80A stack. The firmware program store controls a dedicated Z-80A microprocessor, clocked at 4 MHz, to manage all interface functions. GPIBB connector J2 provides the standard GPIB (IEEE Std 488-1978) interface, and connector J1 supplies a general purpose TTL parallel I/O port (this port is termed the **External I/O** port throughout the manual). All GPIB interface functions are managed by the Texas Instruments TMS 9914 GPIB Adapter.

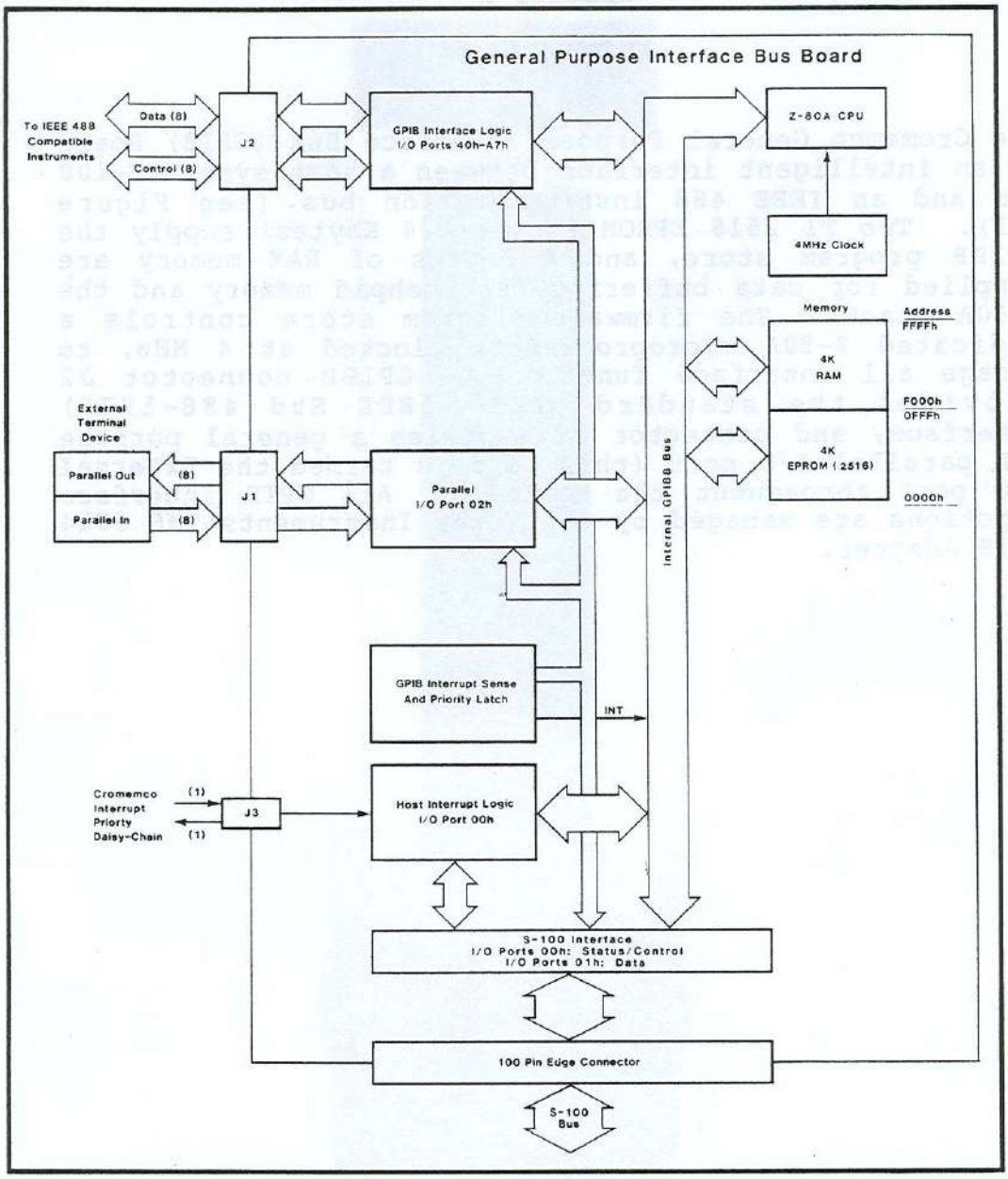


Figure 1-1: GPIBB BLOCK DIAGRAM

The Host communicates with the GPIBB through two S-100 bus parallel I/O ports. DIP switch SW2 locates the GPIBB in the S-100 bus I/O map, while DIP switch SW1 defines the interface's GPIB bus Talk and Listen address.

The GPIBB may issue either non-maskable or maskable vectored interrupt requests to the host processer over the S-100 bus, and the interrupt requests may be coordinated among other requests using Cromemco interrupt priority daisy chain connector J3 on the board.

When enabled under software control, three sources may issue maskable interrupt requests to the GPIBB Z-80A:

1. The host processor through bit **Int GPIBB** of S-100 register **Host Control,**

2. An **External Input** port load strobe, or

3. TMS 9914 GPIB Adapter pin $\overline{\text{INT}}$.

The GPIBB onboard interrupt priorities are software defined. See Chapter 4 for more information on interrupts.

The GPIBB satisfies all requirements of IEEE Std 488-1978, with the exception of the specified maximum data rate. (IEEE Std 488-1978 describes a rather complex interface bus with a large set of uniquely defined signals and states, each with an different acronym. Refer to IEEE Standard Digital Interface for Programmable Instrumentation, 1978, published by **The Institute of Electrical and Electronic Engineers, Inc.,** for a comprehensive description of the standard.) The GPIBB can take the part of the System Controller, a Controller In Charge, a Talker, or a Listener. Although designed to be a dedicated peripheral to an S-100 host system, the GPIBB can also operate as a stand alone Controller, needing only power (+8 volts unregulated) and appropriate EPROM firmware.

Chapter 2

RESET STATE, MEMORY AND HOST I/O

2.1 GPIBB RESET STATE

The GPIBB is reset by any of the following three events:

1. +8 VDC (unregulated) is applied to the GPIBB. In response, GPIBB circuitry generates a momentary active low level Power On Clear (POC) pulse to reset itself.

2. An active low level appears on S-100 bus line RESET , pin 75.

3. A logic 1 followed by a logic 0 is output to bit **Reset** of S-100 register **Host Control.**

Any one of these three events forces the GPIBB to the following state:

1. The GPIBB Z-80A is reset, which means:

   a. Its program counter is reset to 0000h;

   b. Its maskable interrupt request pin is masked (disabled);

   c. Its I and R registers are reset to 00h;

   d. Interrupt mode IM0 is selected.

2. The GPIB Adapter is reset, which means:

   a. All idle states are entered;

   b. All **Serial Poll Register** and **Parallel Poll Register** bits are reset;

   c. All Auxiliary Commands are cleared except **Reset,** which is set, and thus must be cleared during initialization.

3. All eight **External Output** port bits are reset.

4. **Host Status** bits **RDA** and **Int Pending** are reset; bit **TBE** is set.

5. **Host Control** bits **Reset, Int GPIBB, Clr Int** and **Int Msk** are reset.

6. **GPIBB Status** bits **RDA, Ext Int, GPIB Int** and **Host Int** are reset; bit **TBE** is set.

7. **GPIBB Control** bits **NMInt Host Int Msk, GPIB Msk** and **Host Msk** are reset.

8. The **Host Data** and **GPIBB Data** port contents should be considered random until written to for the first time.

2.2 GPIBB MEMORY

The GPIBB is shipped with 4 Kbytes of RAM memory spanning F000h - FFFFh in the GPIBB Z-80A memory map. This memory would typically be used for data buffering, scratchpad memory, and the Z-80A stack. The board also has socket space for two TI 2516 (or equivalent) EPROM memory devices, which supply the Z-80A program store. Socket IC10 (ROM0) spans addresses 0000h - 07FFh (2 Kbytes), and socket IC27 (ROM1) spans 0800h - 0FFFh (2 Kbytes) -- see Figure 2-1. Note the board legend arrows which point to pin 1 of each socket. A GPIBB reset causes the Z-80A to automatically begin program execution at 0000h, so the firmware located in socket ROM0 would typically begin with a GPIBB initialization routine. GPIBB hardware automatically inserts one 250 nSec Wait State during each Z-80A M1 (opcode fetch) cycle, but not during any other cycle type.

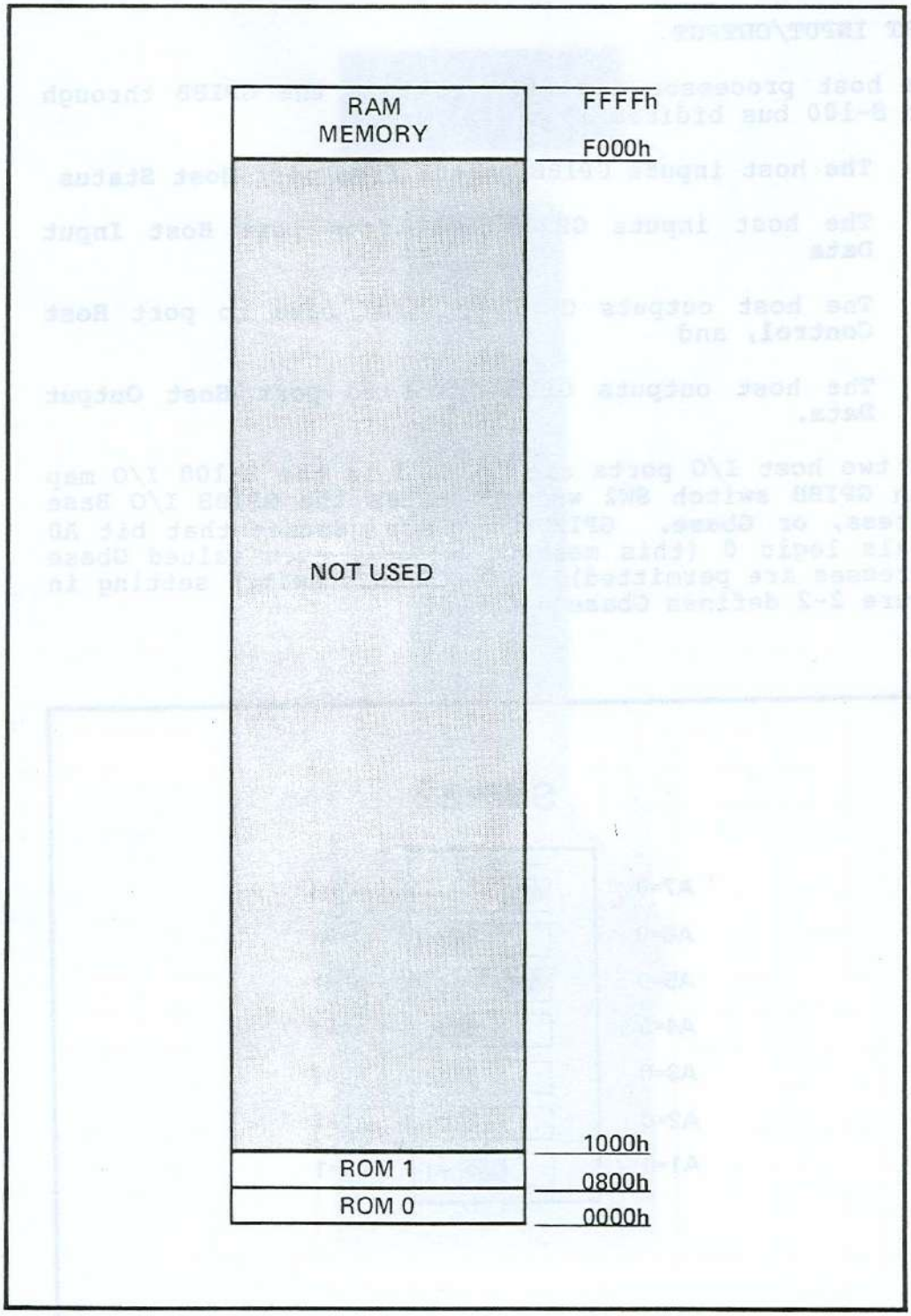


Figure 2-1: GPIBB MEMORY MAP

2.3 HOST INPUT/OUTPUT

The host processor communicates with the GPIBB through two S-100 bus bidirectional I/O ports:

1. The host inputs GPIBB status from port **Host Status**
2. The host inputs GPIBB data from port **Host Input Data**
3. The host outputs GPIBB control bits to port **Host Control**, and
4. The host outputs GPIBB data to port **Host Output Data.**

The two host I/O ports are located in the S-100 I/O map with GPIBB switch SW2 which defines the GPIBB I/O Base Address, or **Gbase.** GPIBB hardware assumes that bit A0 equals logic 0 (this means that only even valued Gbase addresses are permitted). The example switch setting in Figure 2-2 defines Gbase = 5Eh.

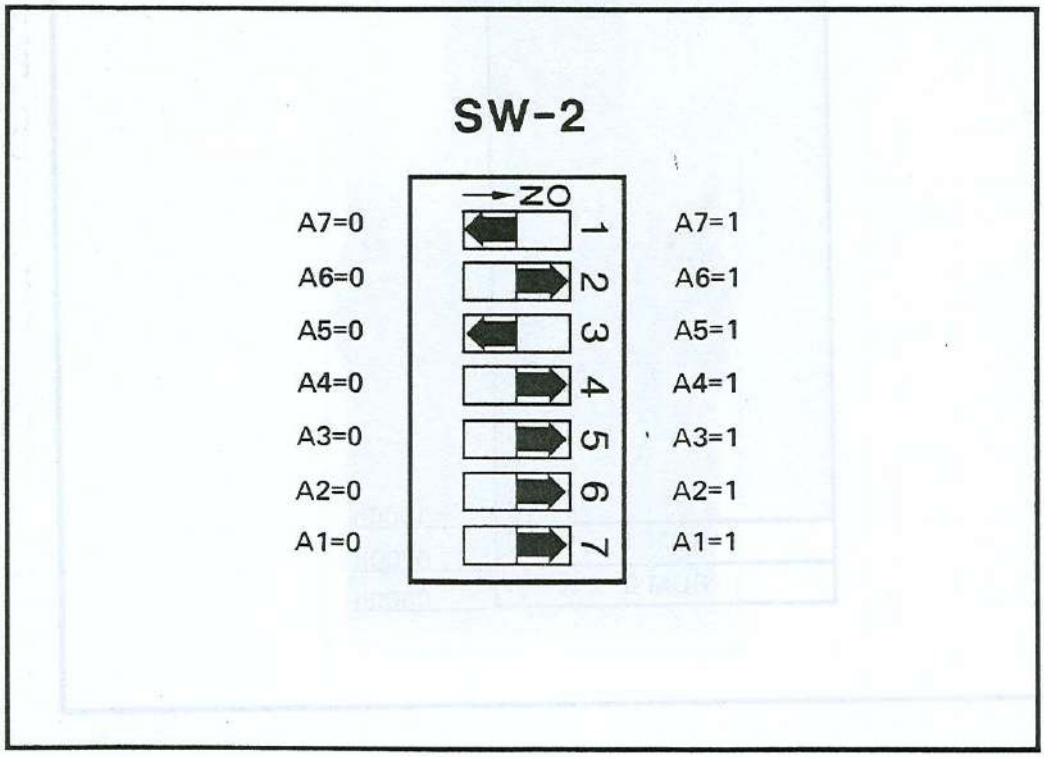


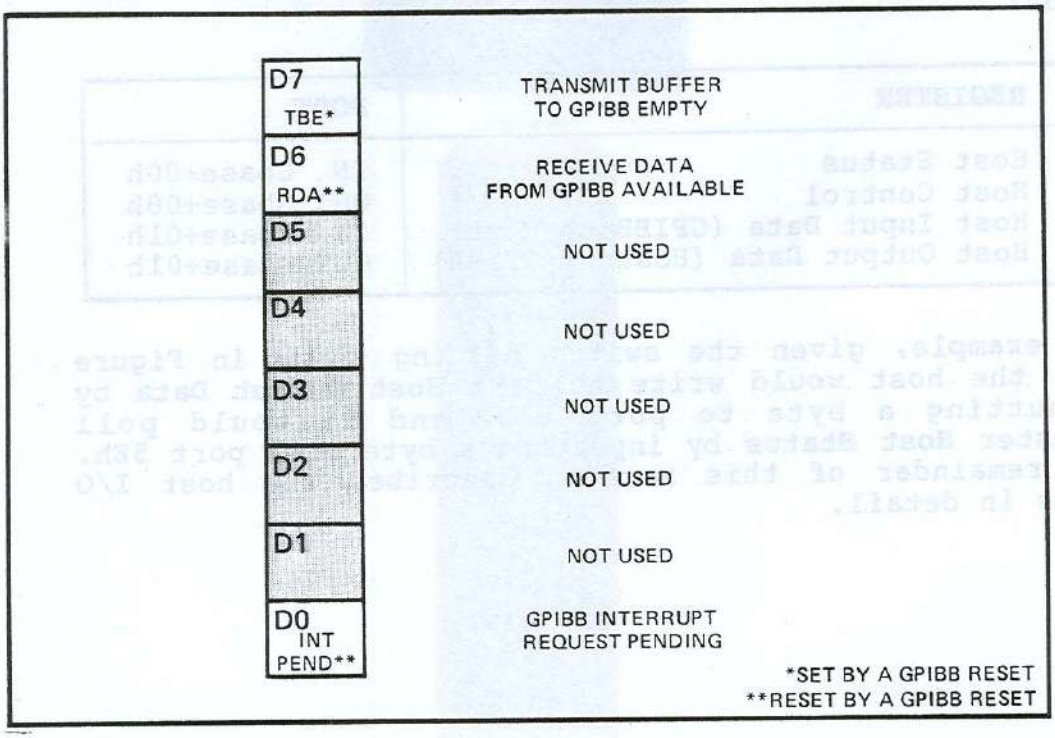
Figure 2-2: GBASE SWITCH SW2

The two host S-100 ports are mapped with offsets of +00h and +01h relative to Gbase. The correspondence is shown below:

REGISTER	PORT
Host Status	IN Gbase+00h
Host Control	OUT Gbase+00h
Host Input Data (GPIBB to Host)	IN Gbase+01h
Host Output Data (Host to GPIBB)	OUT Gbase+01h

For example, given the switch setting shown in Figure 2-2, the host would write to port **Host Output Data** by outputting a byte to port 5Fh, and it would poll register **Host Status** by inputting a byte from port 5Eh. The remainder of this section describes the host I/O ports in detail.

HOST STATUS
IN Gbase+00h



D7 **TBE** -- Transmit Buffer Empty.

Logic 0 => **Host Output Data** is full and cannot be loaded without losing the previous output byte.

Logic 1 => **Host Output Data** is empty, so the host may write another byte to the GPIBB.

D6 **RDA** -- Receive Data Available.

Logic 0 => no data from the GPIBB is available for reading.

Logic 1 => a data byte from the GPIBB is available in the **Host Input Data** register transpose for reading.

D5-D1 Not used.

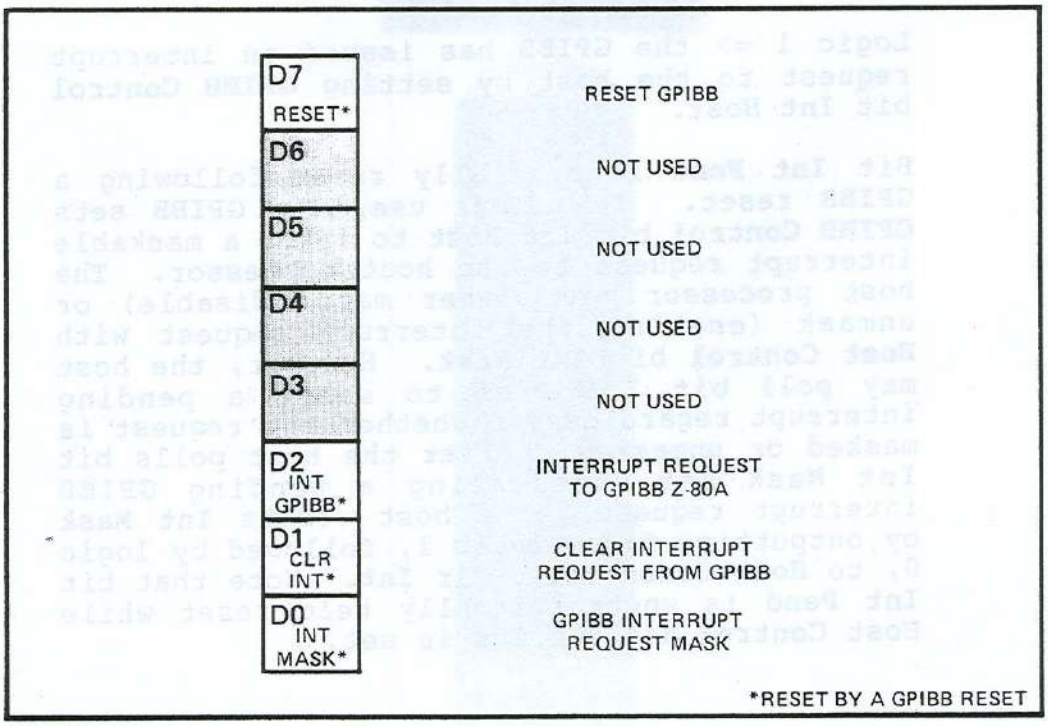
D0 **Int Pend** -- GPIBB Interrupt Request Pending.

Logic 0 => there is no pending GPIBB interrupt request.

Logic 1 => the GPIBB has issued an interrupt request to the host by setting **GPIBB Control** bit **Int Host.**

Bit **Int Pend** is initially reset following a GPIBB reset. In normal use, the GPIBB sets **GPIBB Control** bit **Int Host** to issue a maskable interrupt request to the host processor. The host processor may either mask (disable) or unmask (enable) the interrupt request with **Host Control** bit **Int Mask.** However, the host may poll bit **Int Pend** to sense a pending interrupt regardless of whether the request is masked or unmasked. After the host polls bit **Int Mask** set (indicating a pending GPIBB interrupt request), the host resets **Int Mask** by outputting first logic 1, followed by logic 0, to **Host Control** bit **Clr Int.** Note that bit **Int Pend** is unconditionally held reset while **Host Control** bit **Clr Int** is set.

HOST CONTROL
OUT Gbase+00h



D7 **Reset** -- Reset GPIBB.

Logic 0 => no action.

Logic 1 => holds GPIBB in reset state.

Note that this bit **must** be reset (inactive) after being set (active). Otherwise, the GPIBB is held in the reset state.

D6-D3 Not used.

D2 **Int GPIBB** -- Interrupt Request to GPIBB.

Logic 0 => no operation.

Logic 1 => issues an interrupt request to the GPIBB Z-80A.

The interrupt request from a set **Int GPIBB** bit is automatically removed during interrupt acknowledge from the GPIBB Z-80A. Note that any such interrupt request issued by the host

may be either masked or unmasked by **GPIBB Control** bit **Host Mask.**

D1 **Clr Int** -- Clear Interrupt Request from GPIBB.

Logic 0 => no action.

Logic 1 => clears any pending interrupt request which the GPIBB has issued by setting **GPIBB Control** bit **Int Host,** and inhibits any subsequent requests while bit **Clr Int** remains set.

This bit is initially reset following a GPIBB reset. In normal use, the host interrupt service routine first sets bit **Clr Int,** and then resets it, to remove a GPIBB interrupt request, and to reset **Host Status** bit **Int Pend.**

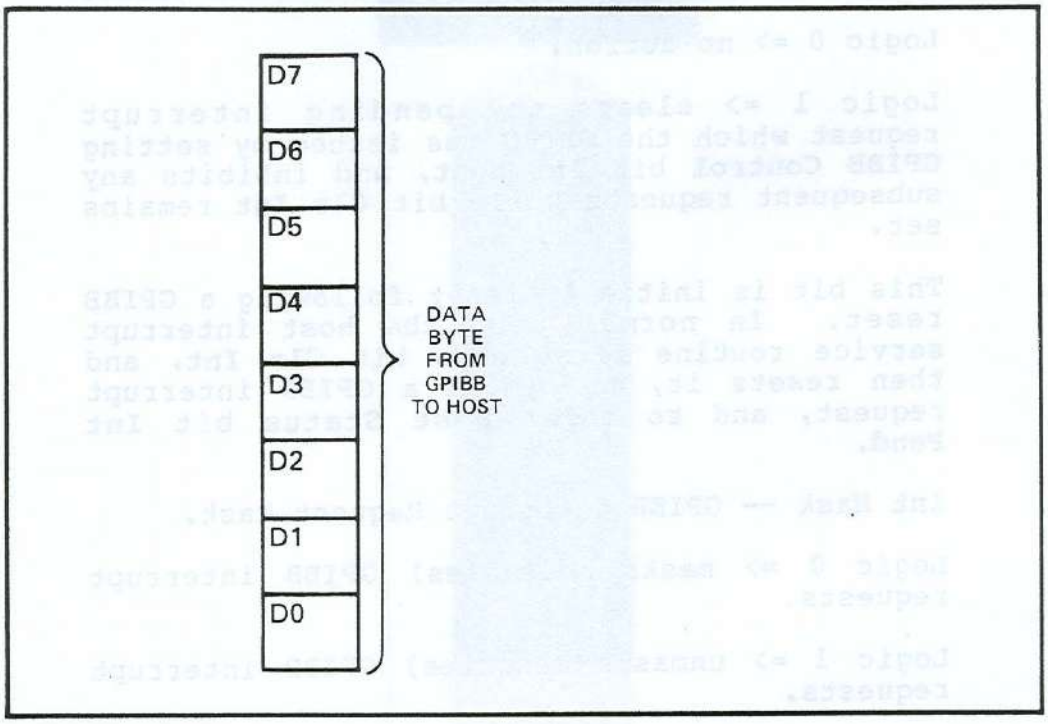
D0 **Int Mask** -- GPIBB Interrupt Request Mask.

Logic 0 => masks (disables) GPIBB interrupt requests.

Logic 1 => unmasks (enables) GPIBB interrupt requests.

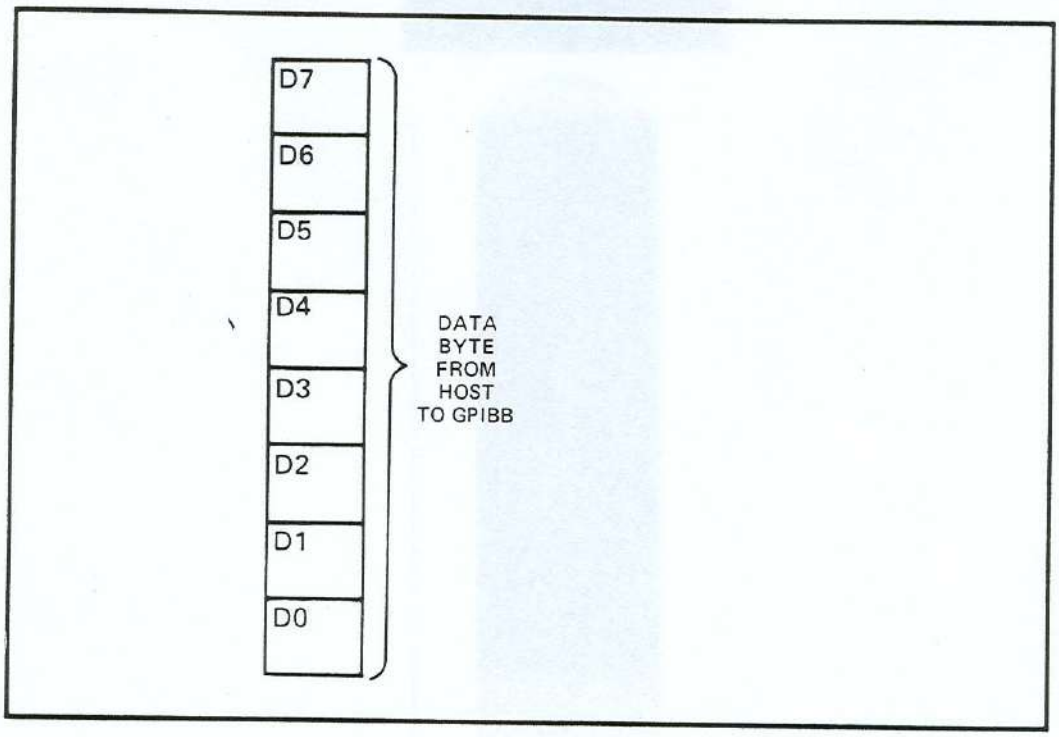
The GPIBB cannot drive S-100 bus line $\overline{\text{INT}}$ active low while bit **Int Mask** is reset. If bit **Int Mask** is set, then the GPIBB can drive S-100 bus line $\overline{\text{INT}}$ active low, provided **Host Control** bit **Clr Int** is reset. S-100 bus line $\overline{\text{INT}}$ goes active low as soon as bit **Int Mask** is set if a GPIBB interrupt request is pending (**GPIBB Control** bit **Host Int** is set). That is, bit **Int Mask** can hold off GPIBB interrupt requests, but it cannot clear them.

HOST INPUT DATA
IN Gbase+01h



This register buffers the data bytes the GPIBB writes to the host. **Host Status** bit **RDA** is set by outputting a byte to its **GPIBB Output Data** register as the GPIBB loads this register. **RDA** is reset as the host reads this register.

HOST OUTPUT DATA
OUT Gbase+01h



This register buffers the data bytes the host writes to the GPIBB. **Host Status** bit **TBE** is reset as the host writes to this register and **TBE** is set by inputting a byte from its **GPIBB Input Data** register as the GPIBB reads this register.

Chapter 3

GPIBB INPUT/OUTPUT

Most internal GPIBB functions are mapped into I/O ports for GPIBB Z-80A access. These functions fall into three general categories.

1. Host communication
2. Interrupt initialization and response
3. TMS 9914 GPIB Adapter communication

Table 3-1 summarizes the GPIBB internal I/O port assignments.

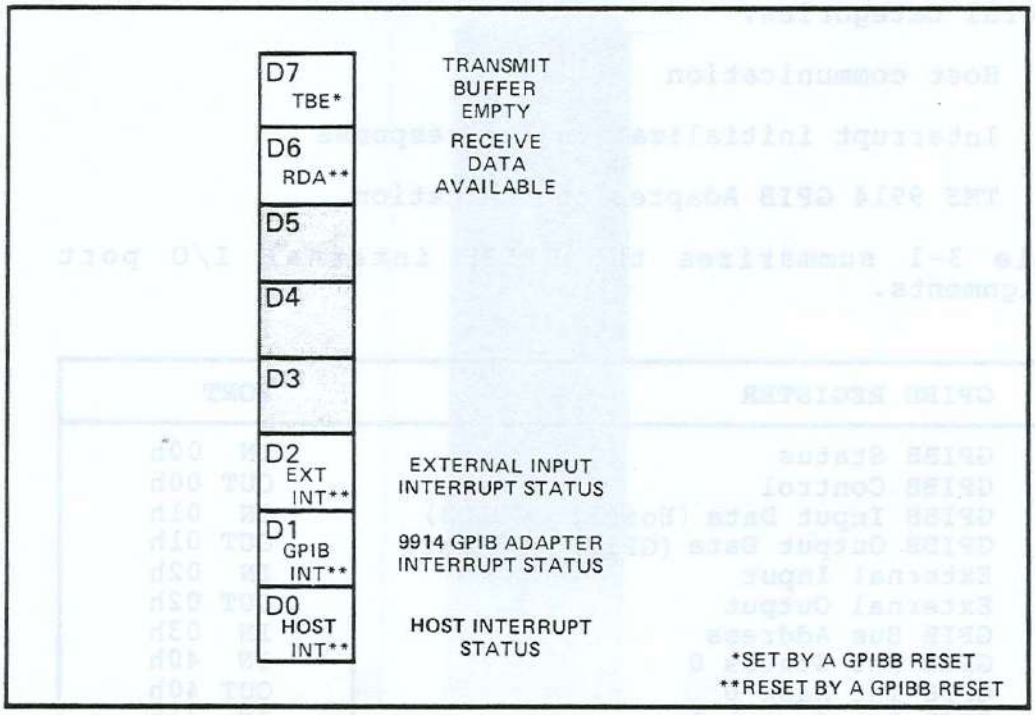
GPIBB REGISTER	PORT
GPIBB Status	IN 00h
GPIBB Control	OUT 00h
GPIBB Input Data (Host to GPIBB)	IN 01h
GPIBB Output Data (GPIBB to Host)	OUT 01h
External Input	IN 02h
External Output	OUT 02h
GPIB Bus Address	IN 03h
GPIB Int Status 0	IN 40h
GPIB Int Mask 0	OUT 40h
GPIB Int Status 1	IN 41h
GPIB Int Mask 1	OUT 41h
GPIB Address Status	IN 42h
GPIB Bus Status	IN 43h
GPIB Auxiliary Commands	OUT 43h
GPIB Address Register	OUT 44h
GPIB Serial Poll	OUT 45h
GPIB Command Pass Through	IN 46h
GPIB Parallel Poll	OUT 46h
GPIB Data Input	IN 47h
GPIB Data Output	OUT 47h

Table 3-1: GPIBB INTERNAL I/O REGISTERS

Section 3.1 of this chapter discusses I/O ports 00h - 03h. These ports do not access LSI device internal registers. Section 3.2 discusses TMS 9914 GPIB Adapter mapped I/O registers, ports 40h - 47h.

3.1 NON-LSI MAPPED PORTS

GPIBB STATUS
IN 00h



D7 **TBE** -- Transmit Buffer Empty.

Logic 0 => **GPIBB Output Data** is full and cannot be loaded without losing the previous output byte.

Logic 1 => **GPIBB Output Data** is empty, so the GPIBB may write another byte to the host.

D6 **RDA** -- Receive Data Available.

Logic 0 => no data from the host is available for reading.

Logic 1 => a data byte from the host is available for reading in register **GPIBB Input Data.**

D5-D3 Not used.

D2 **Ext Int** -- **External Input** Interrupt Status.

Logic 0 => no unmasked **External Input** interrupt request is pending.

Logic 1 => **GPIBB Control** bit **Input Mask** is set and data has been strobed into the **External Input** port; this drives GPIBB Z-80A pin $\overline{\text{INT}}$ active low.

Bit **Ext Int** is typically polled by the GPIBB interrupt service routine to determine the interrupt source. Bit **Ext Int** is automatically reset after register **GPIBB Status** is read.

D1 **GPIB Int** -- TMS 9914 Interrupt Status.

Logic 0 => no unmasked interrupt request from the TMS 9914 is pending.

Logic 1 => **GPIBB Control** bit **GPIB Mask** is set and the TMS 9914 has issued a maskable interrupt request by driving GPIBB Z-80A pin $\overline{\text{INT}}$ active low.

Bit **GPIB Int** is typically polled by the GPIBB interrupt service routine to determine the interrupt source. Bit **GPIB Int** is automatically reset after port **GPIBB Status** is read.

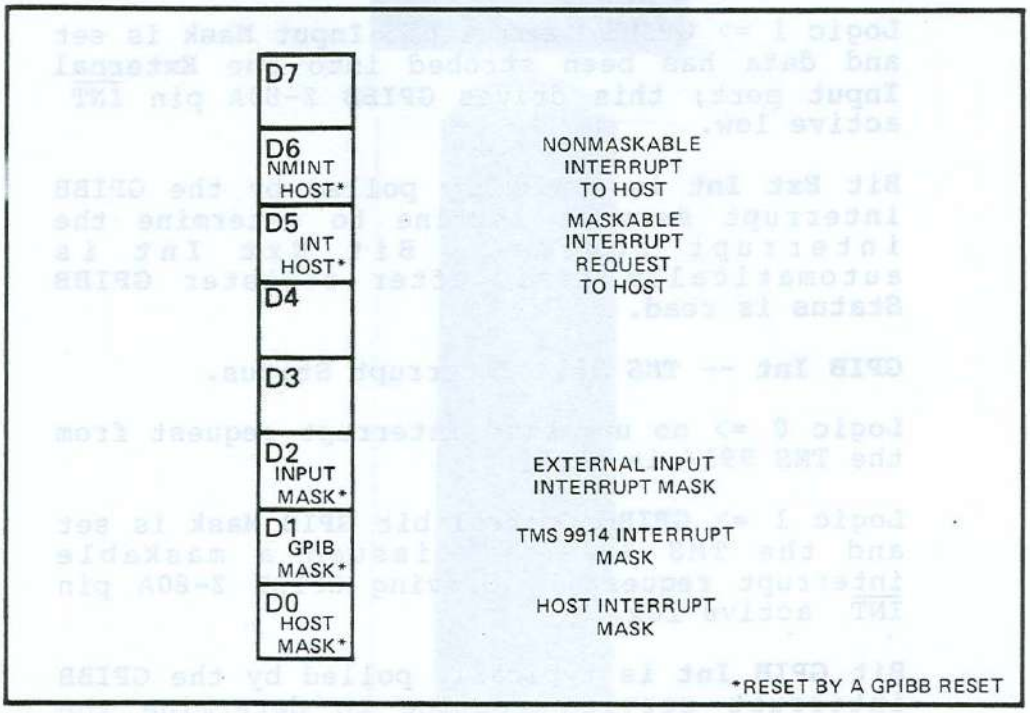
D0 **Host Int** -- Host Interrupt Status.

Logic 0 => no unmasked interrupt request from the host is pending.

Logic 1 => **GPIBB Control** bit **Host Mask** is set and the host has set **Host Control** bit **Int GPIBB**; this drives GPIBB Z-80A pin $\overline{\text{INT}}$ active low.

Bit **Host Int** is typically polled by the GPIBB interrupt service routine to determine the interrupt source. Bit **Host Int** is automatically reset after port **GPIBB Status** is read.

GPIBB CONTROL
OUT 00h



D7 Not used.

D6 **NMInt Host** -- Non-maskable Interrupt To Host.

Logic 0 => no action.

Logic 1 => drives S-100 bus line $\overline{\text{NMI}}$ active low on the next falling edge of S-100 bus signal sM1. The active low level is automatically removed on the following sM1 falling edge. S-100 bus signal sM1 is high during the opcode fetch cycle of each instruction executed.

D5 **Int Host** -- Interrupt Request To Host.

Logic 0 => no action.

Logic 1 => issues a maskable interrupt request to the host. The request drives S-100 bus line $\overline{\text{INT}}$ active low only if **Host Control** bit **Int Mask** is set AND **Host Control** bit **Clr Int** is reset.

D4-D3 Not used.

D2 **Input Mask** -- External Input Interrupt Mask.

Logic 0 => disables **External Input** port interrupts.

Logic 1 => enables **External Input** port interrupts.

When bit **Input Mask** is set, then strobing data into port **External Input** drives GPIBB Z-80A pin $\overline{INT}$ active low. The interrupt request is automatically removed during interrupt acknowledge. If data is strobed into the **External Input** port while bit **Input Mask** is reset, and **Input Mask** is subsequently set, then no interrupt request results.

D1 **GPIB Mask** -- TMS 9914 Interrupt Mask.

Logic 0 => disables TMS 9914 interrupt requests to the GPIBB Z-80A.

Logic 1 => enables TMS 9914 interrupt requests to the GPIBB Z-80A.

When bit **GPIB Mask** is set, an interrupt request from an active low level on TMS 9914 output pin $\overline{INT}$ drives GPIBB Z-80A input pin $\overline{INT}$ active low. The interrupt request is automatically removed during interrupt acknowledge. If TMS 9914 pin $\overline{INT}$ goes low while bit **GPIB Mask** is reset, and **GPIB Mask** is subsequently set, then no interrupt request results.

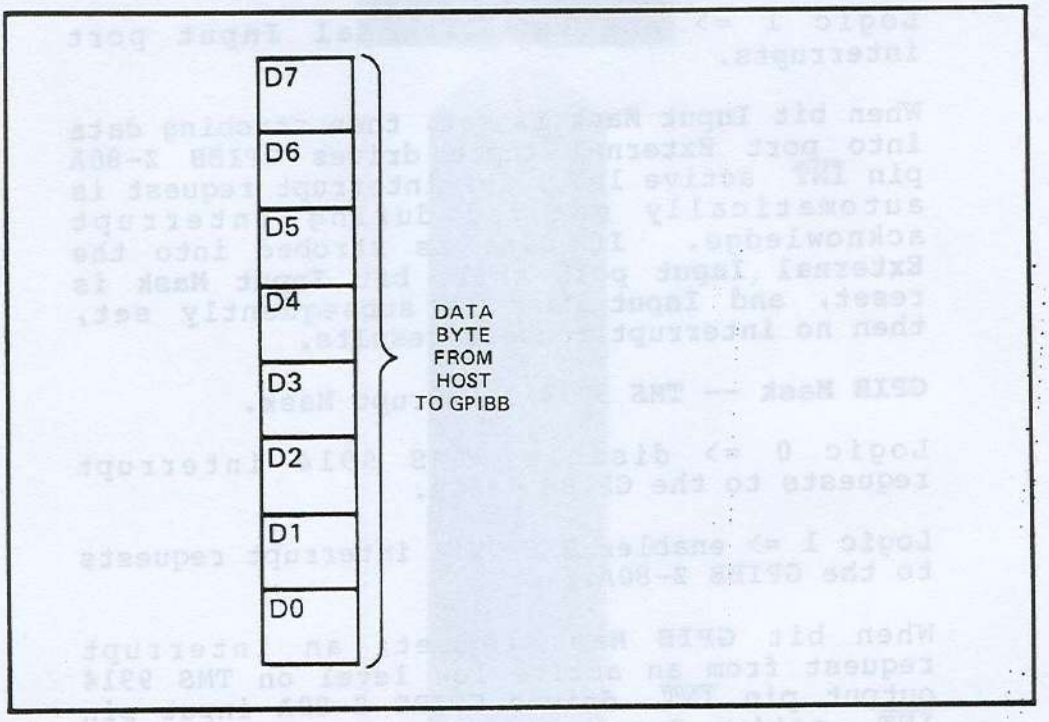
D0 **Host Mask** -- Host Interrupt Mask.

Logic 0 => disables host interrupt requests to the GPIBB Z-80A.

Logic 1 => enables host interrupt requests to the GPIBB Z-80A.

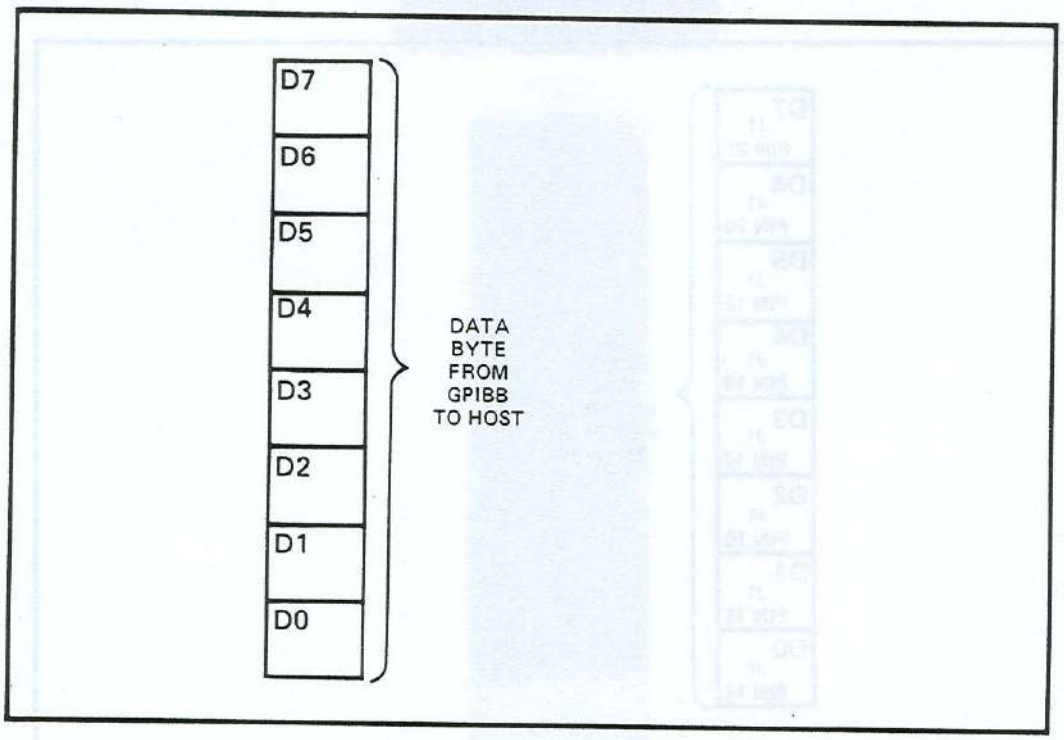
When bit **Host Mask** is set, then GPIBB Z-80A pin $\overline{INT}$ is driven active low when the host sets **Host Control** bit **Int GPIBB**. The interrupt request is automatically removed during interrupt acknowledge from the GPIBB Z-80A. If **Host Control** bit **Int GPIBB** is set while bit **Host Mask** is reset, and **Host Mask** is subsequently set, then no interrupt request results.

GPIBB INPUT DATA
IN 01h



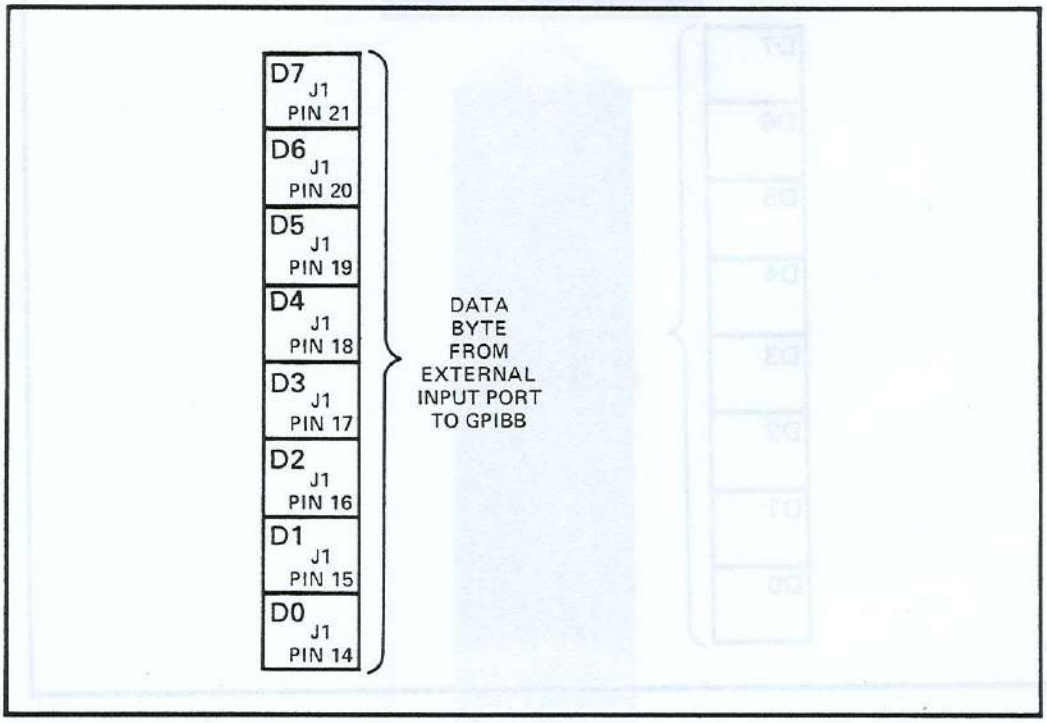
This register buffers the data bytes the host writes to the GPIBB. **GPIBB Status** bit **RDA** is set by outputting a byte to its **Host Output Data** register as the host loads this register. **RDA** is reset as the GPIBB reads this register.

GPIBB OUTPUT DATA
OUT 01h



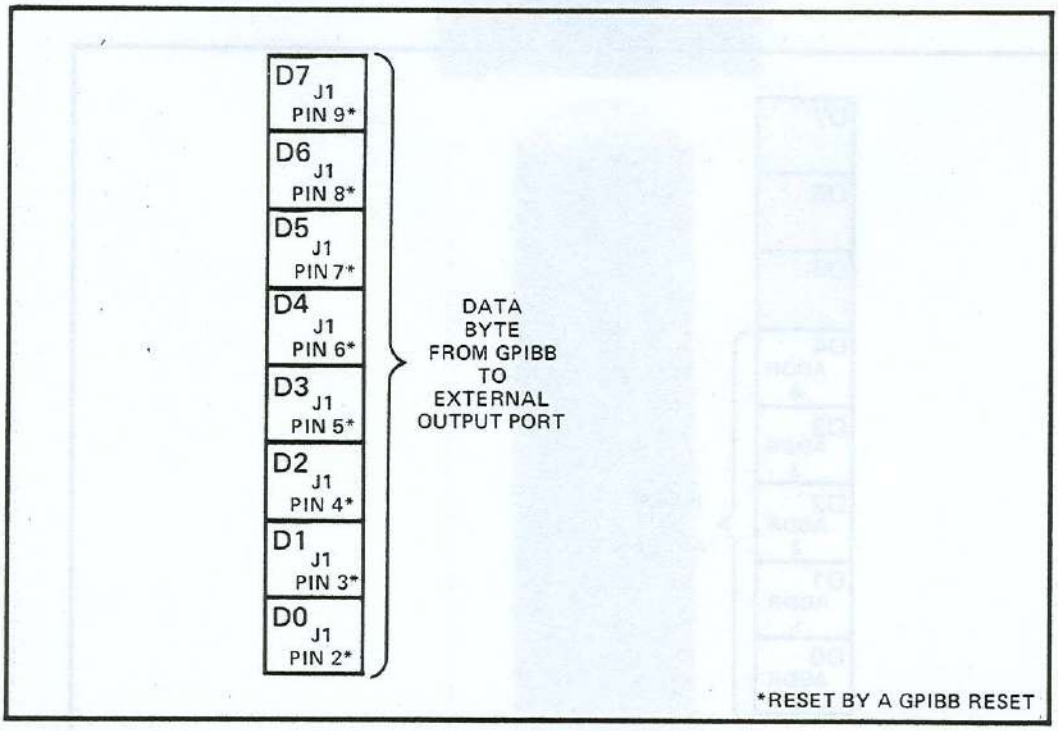
This register buffers the data bytes the GPIBB writes to the host. **GPIBB Status** bit **TBE** is reset as the GPIBB writes to this register, and **TBE** is set by inputting a byte from its **Host Input Data** register as the host reads this register.

EXTERNAL INPUT
IN 02h



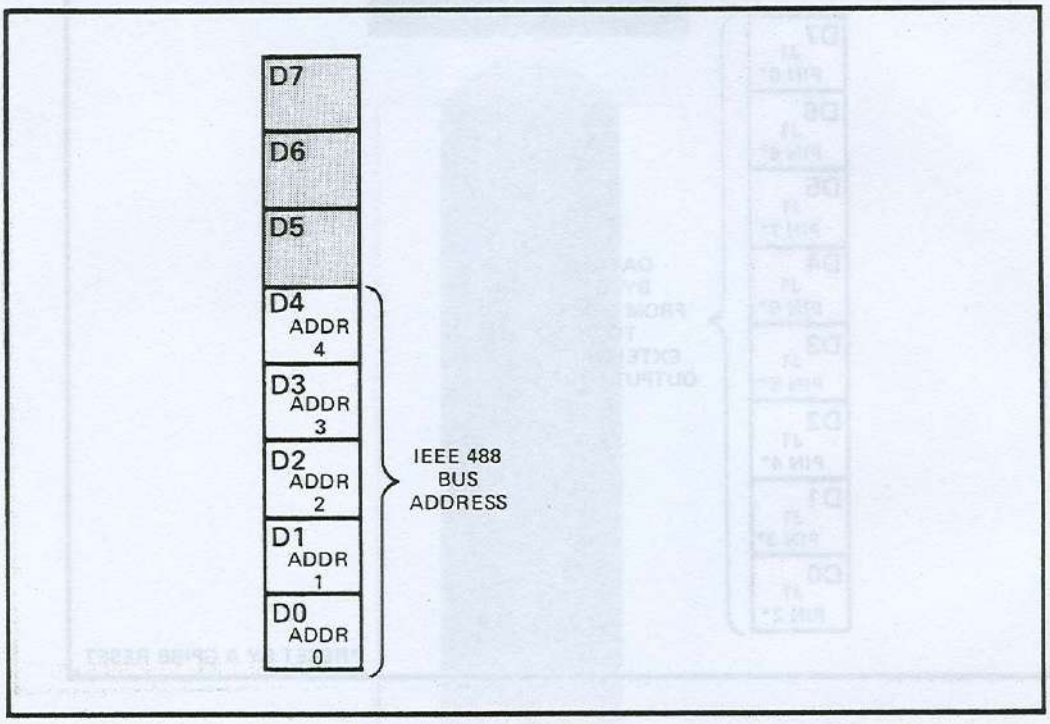
Eight bits of parallel noninverted data are input from GPIBB connector J1 through this port. An external data source strobes a byte into this port by pulsing J1 pin 23, GATE DATA, momentarily high. The rising edge of this strobe may be armed to issue a GPIBB Z-80A interrupt request by setting **GPIBB Control** bit **Input Mask.** If GATE DATA is held high, then the GPIBB may sample the **External Input** data lines in real time. J1 pin 24, $\overline{\text{READ STROBE}}$, pulses low as the GPIBB Z-80A inputs the port data. J1 pin 11, $\overline{\text{WAIT}}$, may be driven low to make the GPIBB Z-80A idle in the Wait State while transferring the data.

EXTERNAL OUTPUT
OUT 02h



The GPIBB outputs eight bits of parallel noninverted data to connector J1 through this port. An external data sink is informed that data has been written to this port by a low pulse on J1 pin 10, $\overline{\text{DATA VALID}}$. J1 pin 11, $\overline{\text{WAIT}}$, may be driven low to make the GPIBB Z-80A idle in the Wait State while transferring the data. All **External Output** bits are reset to logic 0 by a GPIBB reset.

IEEE 488 BUS ADDRESS
IN 03h



This port allows the GPIBB firmware to read the IEEE 488 bus address defined by GPIBB switch SW1. This five bit value is read by the GPIBB Z-80A, and then loaded into port OUT 44h, **GPIB Address Register**, to define the board's primary address (see below). Figure 3-1 illustrates how SW1 would be set to define IEEE 488 bus address 0Dh.

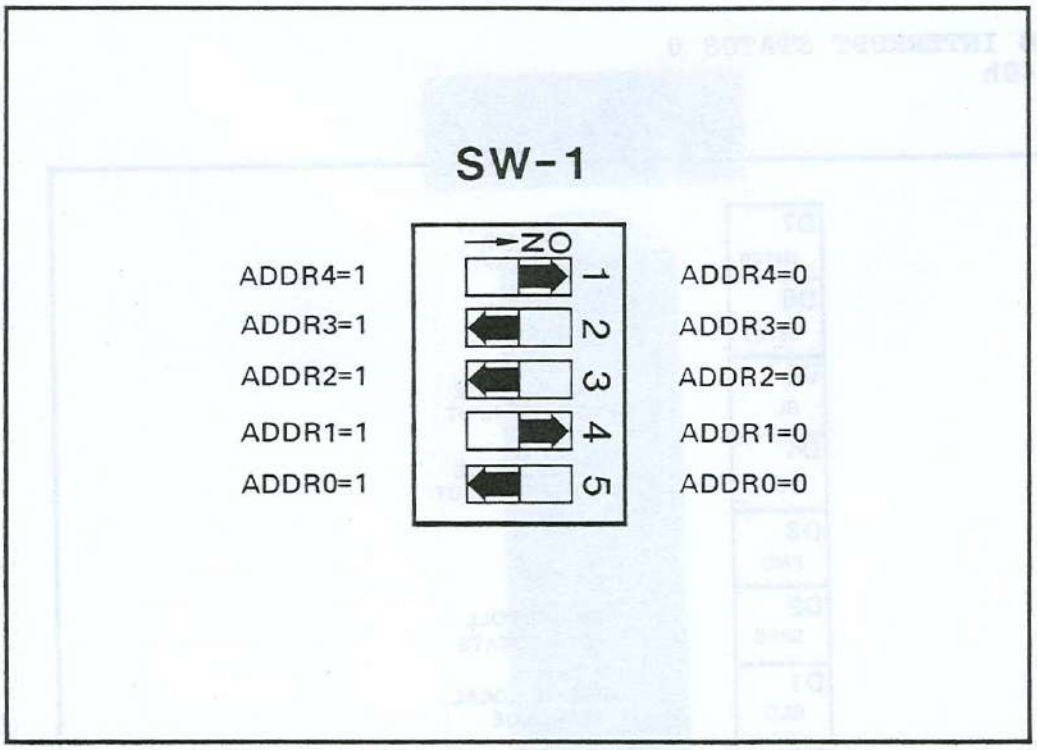
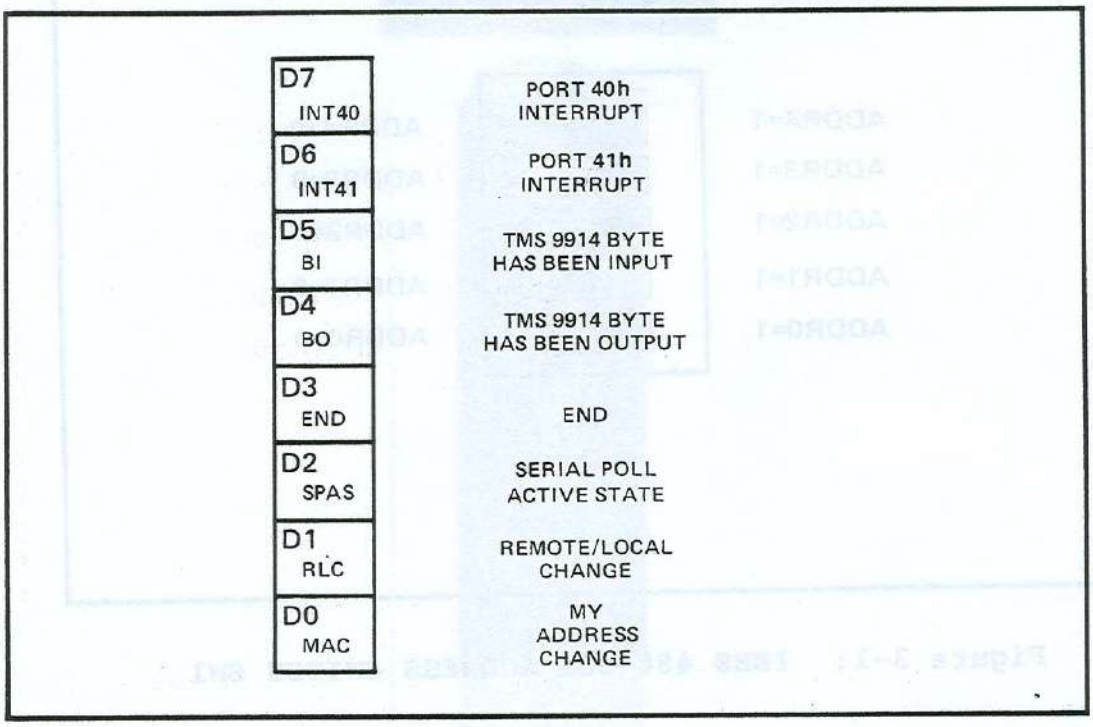


Figure 3-1: IEEE 488 BUS ADDRESS SWITCH SW1

3.2 TMS 9914 GPIB ADAPTER REGISTERS

GPIBB ports 40h through 47h are mapped to access TMS 9914 internal registers. This section presents register descriptions condensed from Texas Instrument's TMS 9914 GPIB Adapter Preliminary Data Manual, 1979.

GPIB INTERRUPT STATUS 0
IN 40h



Status bits D5 - D0 are latched set as their corresponding source conditions occur, regardless of whether they are masked or unmasked by bits of register **GPIB Interrupt Mask 0.** Status bit D7, **INT40**, is latched set whenever one or more unmasked (enabled) bits in the D5 - D0 group of register **GPIB Interrupt Status 0** are set. Likewise, status bit D6, **INT41**, is latched set whenever one or more unmasked (enabled) bits of register **GPIB Interrupt Status 1** are set. Status bits D7 and D5 - D0 are reset when register **GPIB Interrupt Status 0** is read, while bit D6 is reset when port **GPIB Interrupt Status 1** is read. TMS 9914 output pin $\overline{INT}$ is active low whenever (**INT40**) OR (**INT41**) is true.

D7 **INT40** -- Port 40h Interrupt. One or more unmasked (enabled) interrupt sources of register 40h has become active.

D6 **INT41** -- Port 41h Interrupt. One or more unmasked (enabled) interrupt sources of register 41h has become active.

D5 **BI** -- a TMS 9914 Byte has been Input.

D4 **BO** -- a TMS 9914 Byte has been Output, or the TMS 9914 is ready to accept the first data byte.

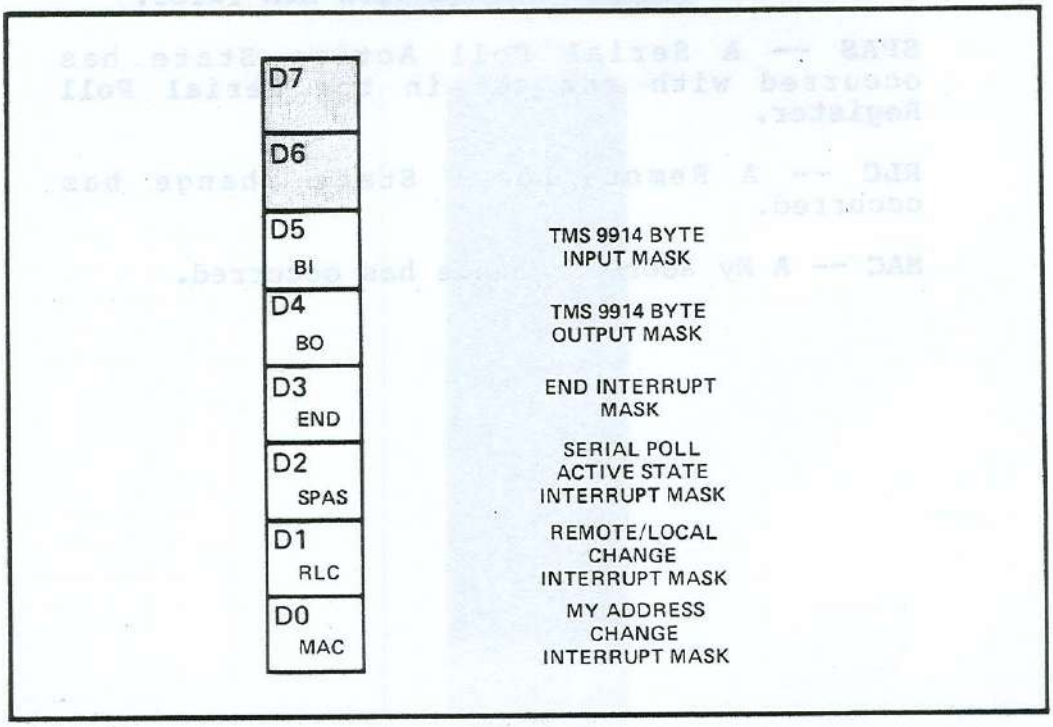
D3 **END** -- an **EOI** has occurred with **ATN** false.

D2 **SPAS** -- A Serial Poll Active State has occurred with **rsv** set in the **Serial Poll Register.**

D1 **RLC** -- A Remote/Local State change has occurred.

D0 **MAC** -- A My Address Change has occurred.

GPIB INTERRUPT MASK 0
OUT 40h



Bits of this register are set (logic 1) to unmask (enable) the corresponding interrupt source condition to drive TMS 9914 output pin $\overline{INT}$ active low. Bits are reset (logic 0) to mask (disable) the corresponding interrupt source.

D7-D6 Not used.

D5 **BI** -- enable interrupt on Byte Input.

D4 **BO** -- enable interrupt on Byte Output.

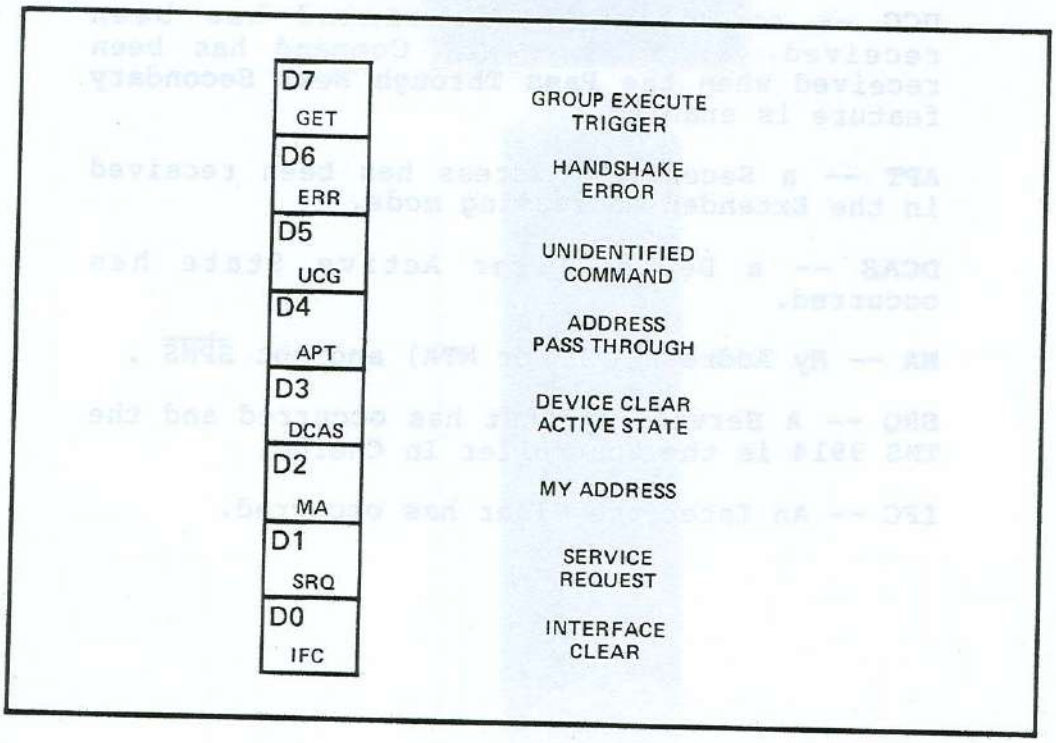
D3 **END** -- enable interrupt on **EOI** with **ATN** false.

D2 **SPAS** -- enable interrupt on Serial Poll Active State.

D1 **RLC** -- enable interrupt on Remote/Local Change.

D0 **MAC** -- enable interrupt on My Address Change.

GPIB INTERRUPT STATUS 1
IN 41h



Status bits D7 - D0 are latched set as their corresponding source conditions occur, regardless of whether they are masked or unmasked by bits of register **GPIB Interrupt Mask 1.** **GPIB Interrupt Status 0** bit **INT41** is latched set whenever one or more unmasked (enabled) bits of register **GPIB Interrupt Status 1** are set. Reading register **GPIB Interrupt Status 1** causes all of its eight bits, as well as bit **INT41** of **GPIB Interrupt Status 0**, to be reset. TMS 9914 output pin $\overline{\text{INT}}$ is active low whenever (**INT40**) OR (**INT41**) is true. The **GET, ERR, UCG, APT, DCAS,** and **MA** events cause an Accept Data State (ACDS) holdoff condition if they are unmasked. This allows the GPIBB Z-80A to respond to the interrupt, recognize the cause by reading the **GPIB Interrupt Status 1**, and take appropriate action before completing the handshake by loading the release ACDS holdoff Auxiliary Command (see **Auxiliary Command Register** descriptions below).

D7 **GET** -- a Group Execute Trigger command has been received.

D6 **ERR** -- an incomplete source handshake error has occurred.

D5 **UCG** -- an Unidentified Command has been received, or a Secondary Command has been received when the **Pass Through Next Secondary** feature is enabled.

D4 **APT** -- a Secondary Address has been received in the Extended Addressing mode.

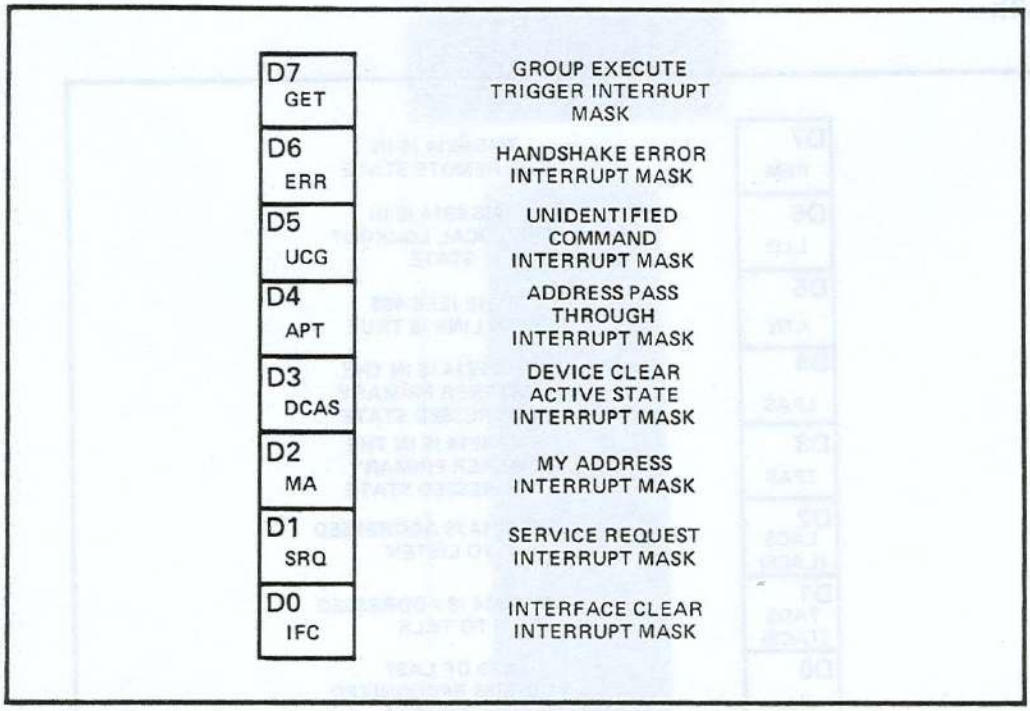
D3 **DCAS** -- a Device Clear Active State has occurred.

D2 **MA** -- My Address (MLA or MTA) and not $\overline{\text{SPMS}}$.

D1 **SRQ** -- A Service Request has occurred and the TMS 9914 is the Controller In Charge.

D0 IFC -- An Interface Clear has occurred.

GPIB INTERRUPT MASK 1
OUT 41h



Bits of this register are set (logic 1) to unmask (enable) the corresponding interrupt source condition to drive TMS 9914 output pin $\overline{INT}$ active low. Bits are reset (logic 0) to mask (disable) the corresponding interrupt source.

D7 **GET** -- enable interrupt on Group Execute Trigger.

D6 **ERR** -- enable interrupt on incomplete source handshake.

D5 **UCG** -- enable interrupt on Unidentified Command or Secondary Command.

D4 **APT** -- enable interrupt on Address Pass Through.

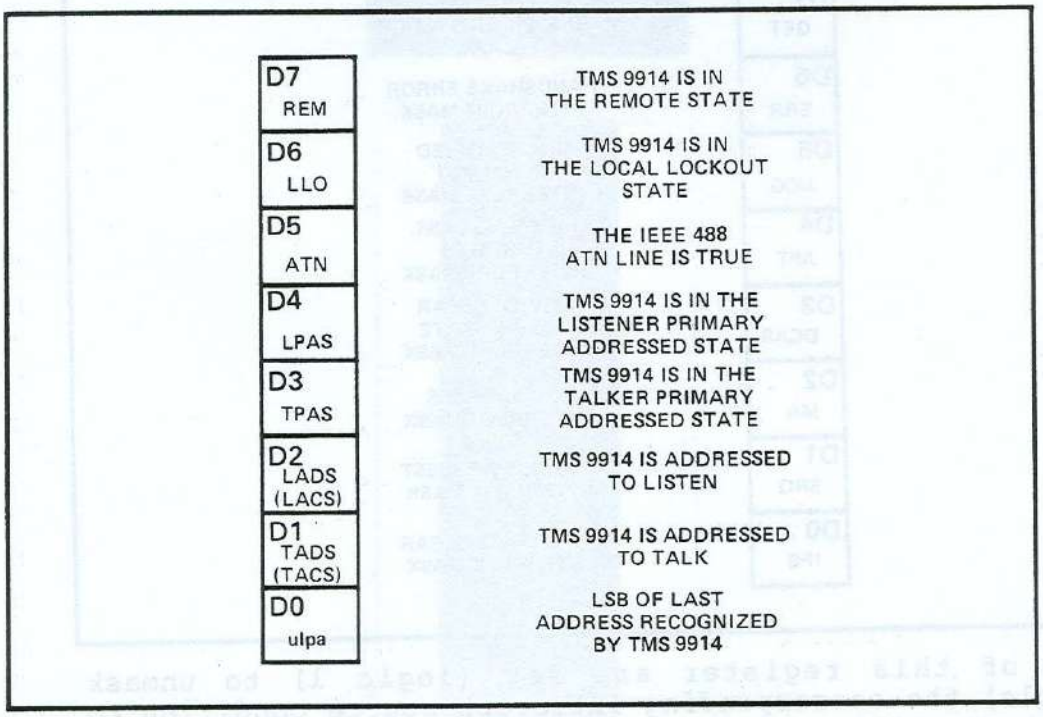
D3 **DCAS** -- enable interrupt on Device Clear Active State.

D2 **MA** -- enable interrupt on My Address.

D1 **SRQ** -- enable interrupt on Service Request.

D0 **IFC** -- enable interrupt on Interface Clear.

GPIB ADDRESS STATE
IN 42h



Status bits read from this register sample the current state of the TMS 9914 internal address logic. Set (logic 1) status bits are interpreted as follows:

D7 **REM** -- the TMS 9914 is in the Remote state.

D6 **LLO** -- the TMS 9914 is in the Local Lockout state.

D5 **ATN** -- IEEE 488 bus line **ATN** is active low.

D4 **LPAS** -- the TMS 9914 is in the Listener Primary Addressed State.

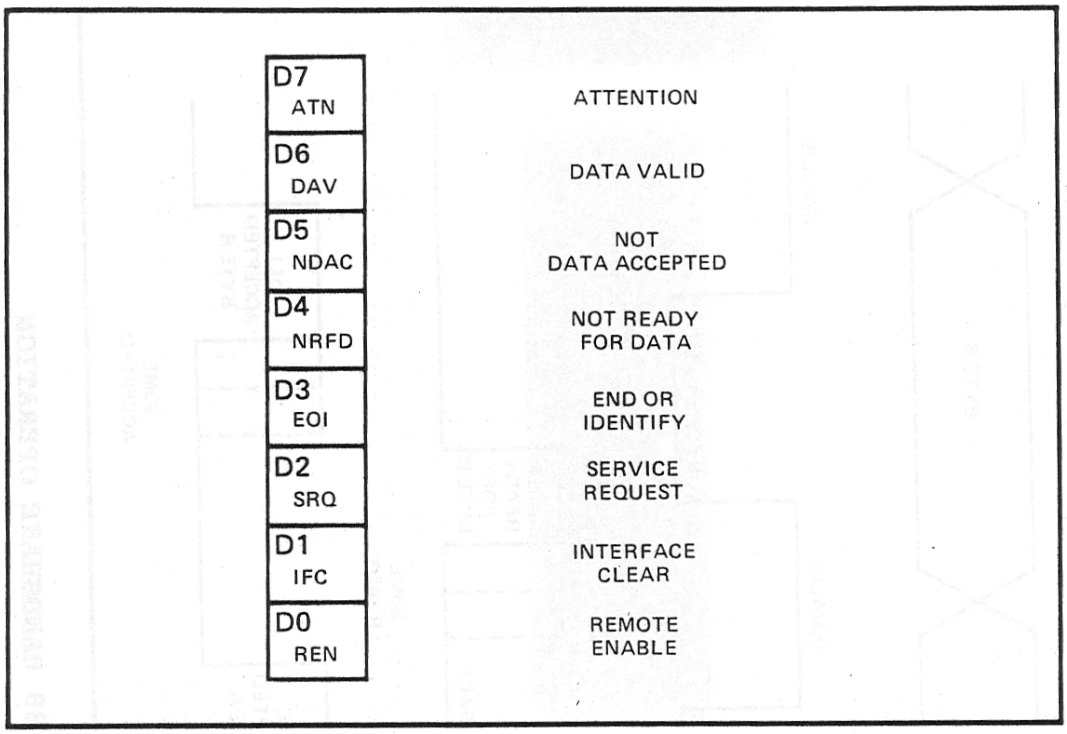
D3 **TPAS** -- the TMS 9914 is in the Talker Primary Addressed State.

D2 **LADS or LACS** -- the TMS 9914 is addressed to Listen.

D1 **TADS or TACS** -- the TMS 9914 is addressed to Talk.

D0 **ulpa** -- the LSB of the last address recognized by the TMS 9914.

GPIB BUS STATUS
IN 43h



The GPIBB Z-80A samples the current IEEE 488 bus management line states by reading this register.

D7 **ATN** -- Attention

D6 **DAV** -- Data Available

D5 **NDAC** -- Not Data Accepted

D4 **NRFD** -- Not Ready For Data

D3 **EOI** -- End Or Identify

D2 **SRQ** -- Service Request

D1 **IFC** -- Interface Clear

D0 **REN** -- Remote Enable

Figure 3-2 illustrates the handshake operation of bus lines **DAV**, **NDAC** and **NRFD** assuming one Talker and several Listeners.

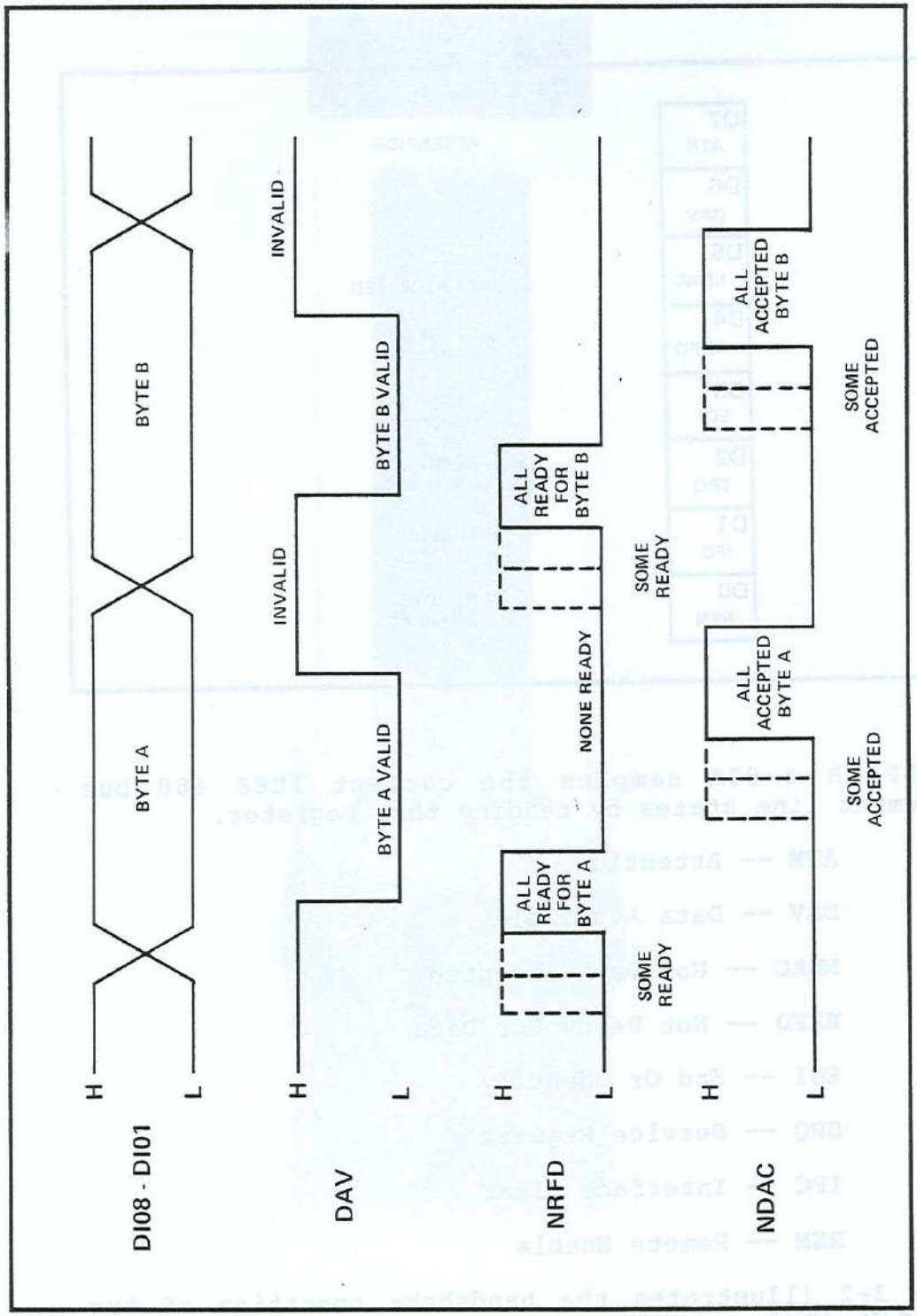
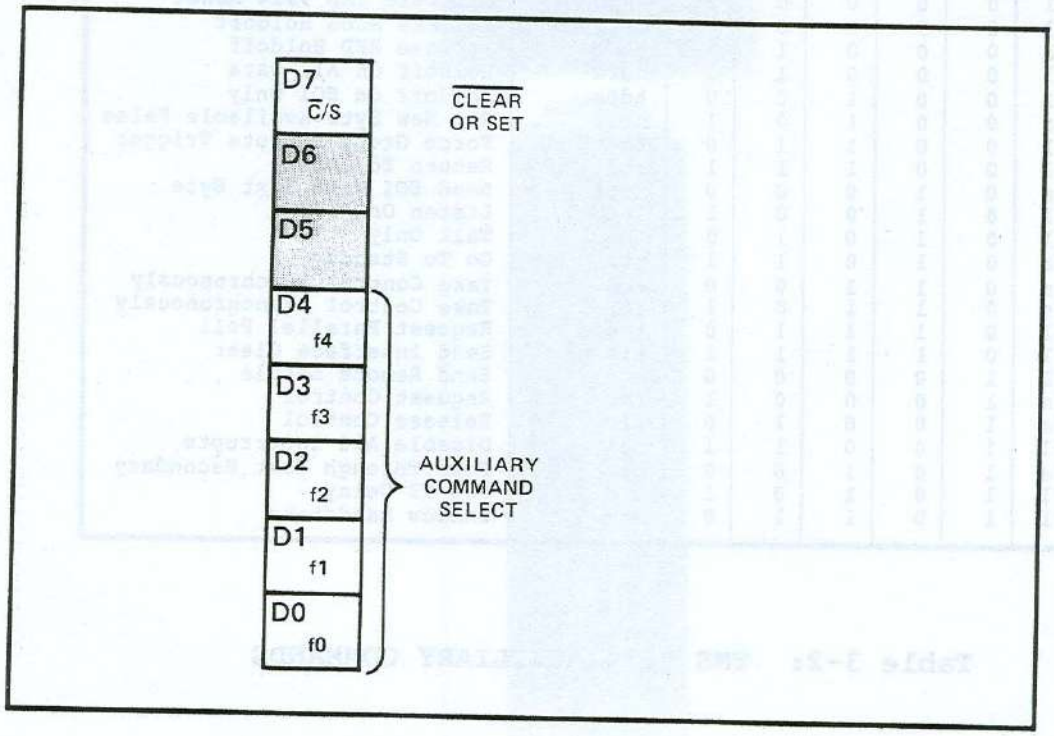


Figure 3-2: IEEE 488 HANDSHAKE OPERATION

AUXILIARY COMMAND REGISTER
OUT 43h



D7 $\overline{C}$ **/S** -- $\overline{\text{Clear}}$ or Set (see text)

D6-D5 Not used

D4-D0 **f4-f0**-- Auxiliary Command Select (see text)

This register controls many of the TMS 9914 special features. The GPIBB Z-80A issues Auxiliary Commands to the TMS 9914 by outputting a byte to this register, and the bits of the byte determine the command issued as shown in Table 3-2 below.

C*/S	f4	f3	f2	f1	f0	MNENONIC	FUNCTION
0/1	0	0	0	0	0	swrst	Software TMS 9914 Reset
0/1	0	0	0	0	1	dacr	Release ACDS Holdoff
n/a	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
n/a	0	0	1	0	1	nbaf	Set New Byte Available False
0/1	0	0	1	1	0	fget	Force Group Execute Trigger
0/1	0	0	1	1	1	rtl	Return To Local
n/a	0	1	0	0	0	feoi	Send EOI With Next Byte
0/1	0	1	0	0	1	lon	Listen Only
0/1	0	1	0	1	0	ton	Talk Only
n/a	0	1	0	1	1	gts	Go To Standby
n/a	0	1	1	0	0	tcs	Take Control Synchronously
n/a	0	1	1	0	1	tca	Take Control Asynchronously
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
n/a	1	0	0	0	1	rqc	Request Control
n/a	1	0	0	1	0	rlc	Release Control
0/1	1	0	0	1	1	dai	Disable All Interrupts
n/a	1	0	1	0	0	pts	Pass Through Next Secondary
0/1	1	0	1	0	1	stdl	Set TI Delay
0/1	1	0	1	1	0	shdw	Shadow Handshake

Table 3-2: TMS 9914 AUXILIARY COMMANDS

A number of the functions are of the Clear/Set type. If a command is issued with the $\overline{C}$ /S bit set, then the function is selected and remains selected until the same command is issued with the $\overline{C}$ /S bit reset. These commands appear with a **0/1** entry in the $\overline{C}$ /S column. The Talk Only (**ton**) and Listen Only (**lon**) commands are of this type. Other commands, such as force EOI (**feoi**) and release RFD holdoff (**rhdf**), have a pulsed mode of operation. These commands appear with an **n/a** (not applicable) entry in the $\overline{C}$ /S column. The Force Group Execute Trigger (**fget**) and Return To Local (**rtl**) commands can operate in either Clear/Set or pulsed modes.

The following paragraphs describe each of the TMS 9914 Auxiliary Commands.

Software Reset (swrst) 0/1 X X 0 0 0 0 0

Writing this command with bit $\overline{C}$ /S set causes all inputs to be ignored and the TMS 9914 returns to the idle state. The **Serial Poll Register** and **Parallel Poll Register** are not cleared, and the following TMS 9914 internal states are selected:

- **SIDS** (source idle state)
- **AIDS** (acceptor idle state)
- **TIDS** (talker idle state)
- **TPIS** (talker primary idle state)
- **LIDS** (listener idle state)
- **LPIS** (listener primary idle state)
- **CIDS** (controller idle state)
- **LOCS** (local state)
- **NPRS** (negative poll response state)
- **PPIS** (parallel poll idle state)
- **SPIS** (serial poll idle state).

This command is active following a TMS 9914 hardware reset. The TMS 9914 may then be configured, but it is held in the idle condition until the **swrst** command is written with bit $\overline{C}$ /S reset.

Release DAC Holdoff (dacr) 0/1 X X 0 0 0 0 1

The Data Accepted (DAC) holdoff provides the GPIBB Z-80A time to respond to unrecognized commands, secondary addresses, device trigger or device clear commands. The holdoff is released by the Z-80A when the required action has been taken. Normally the command is loaded with the $\overline{C}$ /S bit reset. When used with the address pass through feature, however, bit $\overline{C}$ /S is set if the secondary address was valid, or reset if the secondary address was invalid.

Release RFD Holdoff (rhdf) n/a X X 0 0 0 1 0

Releases Ready For Data (RFD) holdoff, caused by a **hdfa** or **hdfe.**

Holdoff On All Data (hdfa) 0/1 X X 0 0 0 1 1

A Ready For Data (RFD) holdoff occurs on every data byte until the **hdfa** command is loaded with $\overline{C}$ /S reset. In such cases, the handshake must be completed by issuing the **rhdf** command.

Holdoff On End (hdfe) 0/1 X X 0 0 1 0 0

An RFD holdoff occurs when an end of data string message (**EOI** true with **ATN** false) is received over the interface. Command **rhdf** releases this holdoff.

Set New Byte Available False (nbaf) n/a X X 0 0 1 0 1

If a Talker is interrupted before the byte stored in the **Data Out Register** is sent across the interface, then the byte is normally transmitted as soon as bus line **ATN** goes false. Issuing command **nbaf** suppresses transmission of this byte if the interrupt makes it unnecessary.

Force Group Execute Trigger (fget) 0/1 X X 0 0 1 1 0

Issuing this command with bit $\overline{C}$ /S reset causes TMS 9914 output pin TRIGGER to pulse high for approximately 5 clock cycles (1.25 uSec). Issuing this command with bit $\overline{C}$ /S set causes pin TRIGGER to go high until another **fget** command is issued with $\overline{C}$ /S reset. No interrupts or handshakes are initiated.

Return To Local (rtl) 0/1 X X 0 0 1 1 1

If the **rtl** command is issued with bit $\overline{C}$ /S reset, then **GPIB Address Status** bit **REM** is reset, but it may be set at any time by a **REN** command from the Controller In Charge. If the **rtl** is issued with bit $\overline{C}$ /S set, then status bit **REM** is reset, and it cannot be again set until the **rtl** command is issued with bit $\overline{C}$ /S reset. The **rtl** command has no effect if the TMS 9914 is in the Local Lockout (LLO) mode.

Force End Or Identify (feoi) n/a X X 0 1 0 0 0

Sends the **EOI** message with the next data byte. The **EOI** line is then reset.

Listen Only (lon) 0/1 X X 0 1 0 0 1

Activates the Listener State until the **lon** command is issued with bit $\overline{C}$ /S reset.

Talk Only (ton) 0/1 X X 0 1 0 1 0

Activates the Talker State until the **ton** command is issued with bit $\overline{C}$ /S reset.

Note that the **ton** and **lon** commands are included for use in systems without a Controller. However, when the TMS 9914 is functioning as a Controller, it uses the **ton** and **lon** commands to set itself up as a Talker or Listener, respectively. Note that these functions must be reset when sending **UNL** or **OTA.**

Go To Standby (gts) n/a X X 0 1 0 1 1

The Controller In Charge issues this command to force bus line **ATN** false.

Take Control Synchronously (tcs) n/a X X 0 1 1 0 0

This command is used by the Controller In Charge to set bus line **ATN** true after any handshake in progress is completed, and so gain control of the interface. If the Controller is not a true Listener, the **shdw** (Shadow Handshake) command must be issued with bit $\overline{C}$ /S set before issuing this command, allowing the Controller to participate in the Listener handshake without exchanging data. **ATN** is forced true at the end of the byte transfer to insure that the data byte is not lost or corrupted.

Take Control Asynchronously (tca) n/a X X 0 1 1 0 1

Immediately forces bus line **ATN** active low. Data loss or corruption may occur if Talker/Listeners are in the process of transferring a data byte.

Request Parallel Poll (rpp) 0/1 X X 0 1 1 1 0

The Controller In Charge issues this command with bit $\overline{C}$ /S set to send the Parallel Poll command over the interface (the TMS 9914 must be in the Controller Active State for line **ATN** to be forced true). The poll is completed by reading the **Command Pass Through Register** to obtain the status bits, then issuing command **rpp** with bit $\overline{C}$ /S reset.

Send Interface Clear (sic) 0/1 X X 0 1 1 1 1

The System Controller issues this command with bit $\overline{C}$ /S set to force bus line **IFC** active low. After observing the IEEE 488 minimum time duration (100 uSec), the System Controller must issue the same command with bit $\overline{C}$ /S reset to force bus line **IFC** inactive high. The System Controller is then put in the Controller Active State.

Send Remote Enable (sre) 0/1 X X 1 0 0 0 0

The System Controller issues this command with bit $\overline{C}$ /S set to force bus line **REN** active low to send the Remote Enable message over the interface. Line **REN** is forced inactive high by issuing the same command with bit $\overline{C}$ /S reset.

Request Control (rqc) n/a X X 1 0 0 0 1

The GPIBB Z-80A issues this command after the **TCT** command has been recognized. The TMS 9914 then waits for bus line **ATN** to go inactive high, then enters the Controller Active State (**CACS**).

Release Control (rlc) n/a X X 1 0 0 1 0

Releases bus line **ATN** after a **TCT** command has been sent passing control to another device.

Disable All Interrupts (dai) n/a X X 1 0 0 1 1

Disables (floats) TMS 9914 interrupt request output pin $\overline{INT}$. Registers **GPIB Interrupt Status 0** and **GPIB Interrupt Status 1** are not affected, nor are any selected holdoffs.

Pass Through Next Secondary (pts) n/a X X 1 0 1 0 0

This command remotely configures a Parallel Poll. The Parallel Poll Configure (**PPC**) message is passed through the TMS 9914 as an unrecognized addressed command, then identified by the GPIBB Z-80A. The **pts** command is then issued and the next byte received (a **PPE**, or Parallel Poll Enable message) is routed through the **Command Pass Through Register.** The GPIBB reads the **PPE** message, and writes it to the **Parallel Poll Register.**

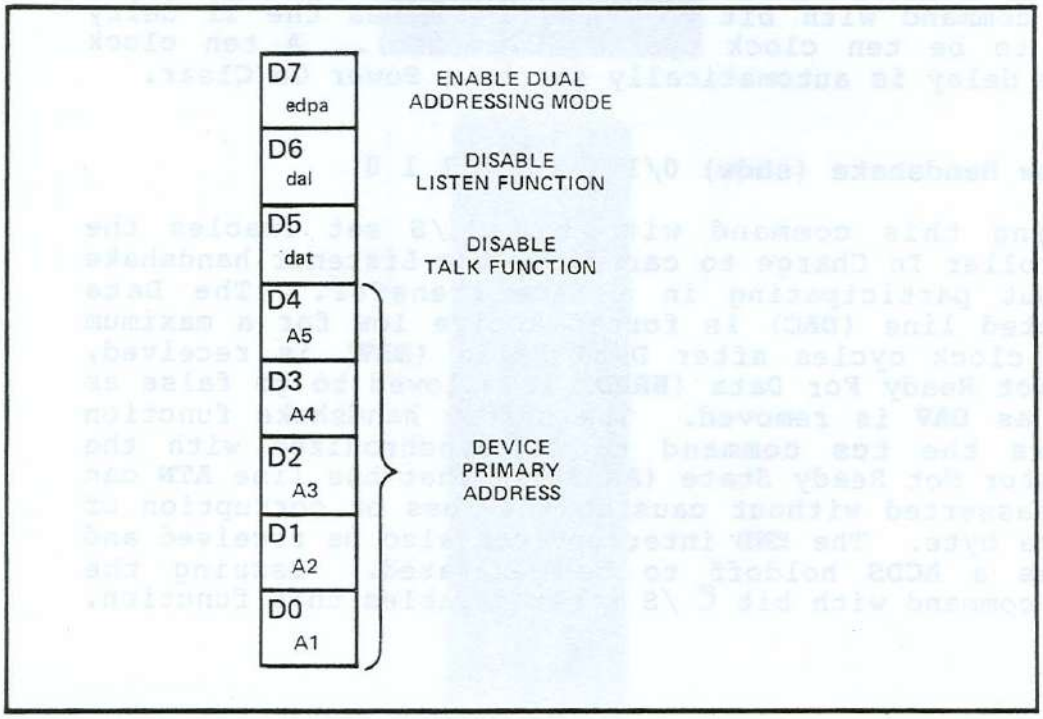
Set T1 Delay (stdl) 0/1 X X 1 0 1 0 1

Issuing this command with bit $\overline{C}$ /S set defines the T1 delay time to be six clock cycles (1.5 uSec). Issuing this command with bit $\overline{C}$ /S reset defines the T1 delay time to be ten clock cycles (2.5 uSec). A ten clock cycle delay is automatically set by a Power On Clear.

Shadow Handshake (shdw) 0/1 X X 1 0 1 1 0

Issuing this command with bit $\overline{C}$ /S set enables the Controller In Charge to carry out the Listener handshake without participating in a data transfer. The Data Accepted line (**DAC**) is forced active low for a maximum of 3 clock cycles after Data Valid (**DAV**) is received, and Not Ready For Data (**NRFD**) is allowed to go false as soon as **DAV** is removed. The shadow handshake function allows the **tcs** command to be synchronized with the Acceptor Not Ready State (**ANRS**) so that bus line **ATN** can be reasserted without causing the loss or corruption of a data byte. The **END** interrupt can also be received and causes a **ACDS** holdoff to be generated. Issuing the **shdw** command with bit $\overline{C}$ /S reset disables this function.

GPIB ADDRESS REGISTER
OUT 44h



D7 **edpa** -- Enable Dual Addressing Mode

D6 **dal** -- Disable Listen Function

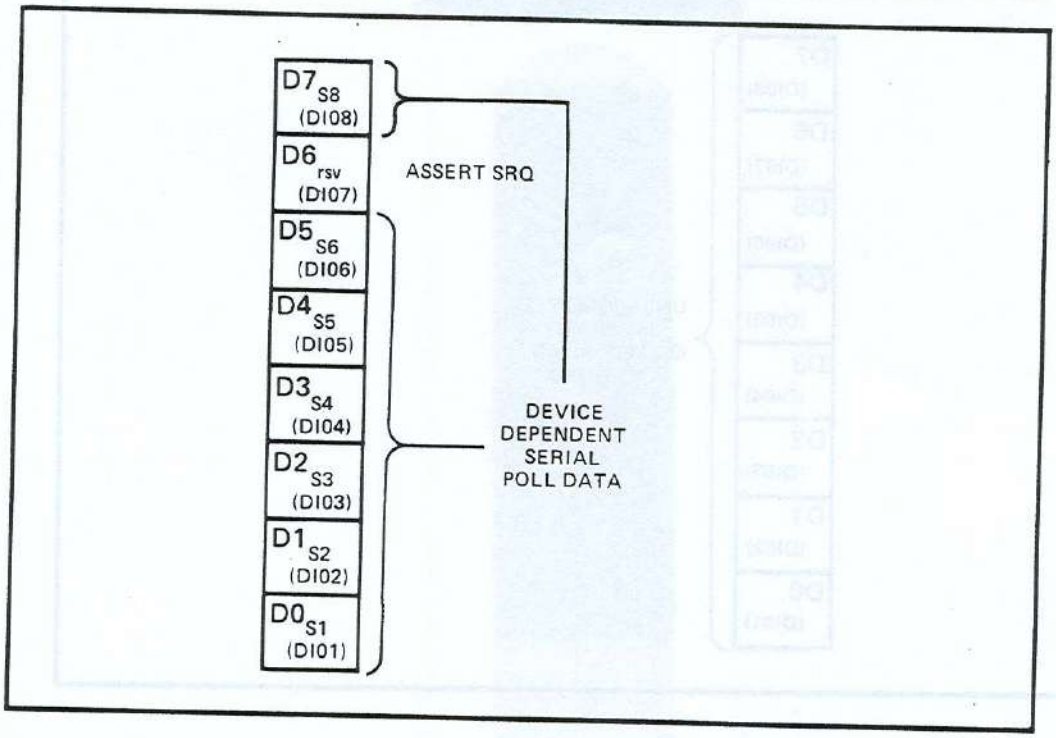
D5 **dat** -- Disable Talk Function

D4-D0 **A5 through A1** -- Device Primary Address

A Power On Clear or a **swrst** command with bit $\overline{C}$ /S set puts the TMS 9914 into an idle state. The **IEEE 488 Bus Address** register (switch SW1) is then read by the GPIBB Z-80A, and written to bits **A5** through **A1** of this register to define the Device Primary Address.

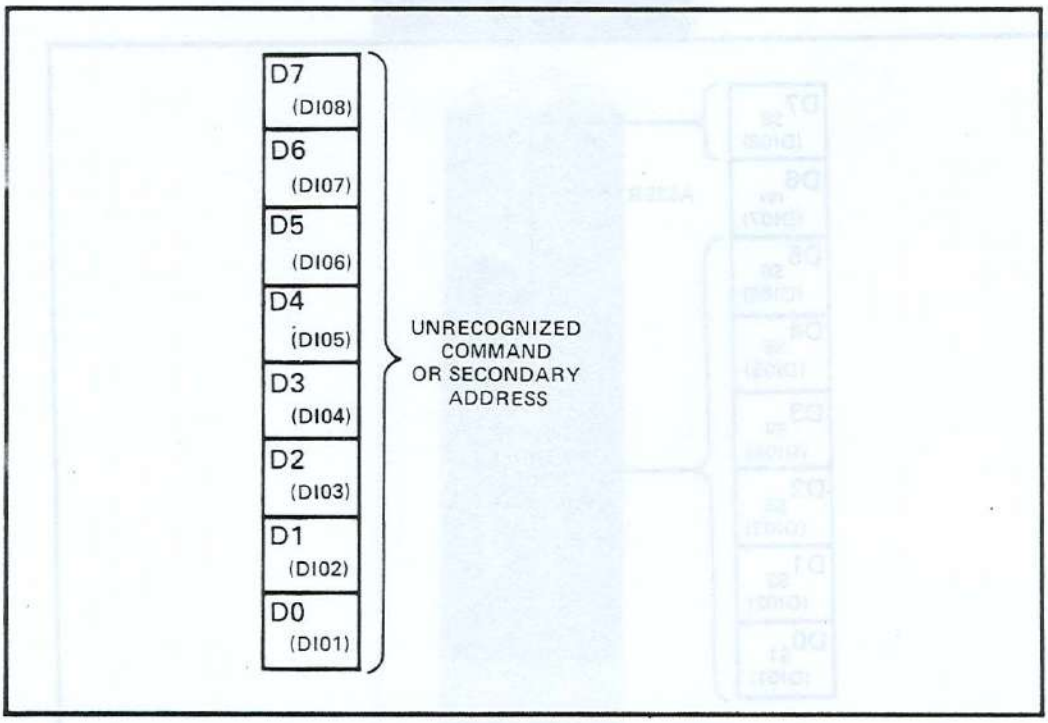
Bit **edpa** is set to enable the TMS 9914 Dual Addressing Mode, and reset to disable it. When enabled, the least significant address bit is ignored by the address comparator, meaning that the TMS 9914 then responds to two consecutive addresses. **GPIB Address Status** bit **ulpa** may then be polled to differentiate between the two addresses. The **dat** and **dal** bits may be set to disable the Talk and Listen functions, respectively. Two separate devices may then use the same primary address if one only talks, and the other only listens.

SERIAL POLL REGISTER
OUT 45h



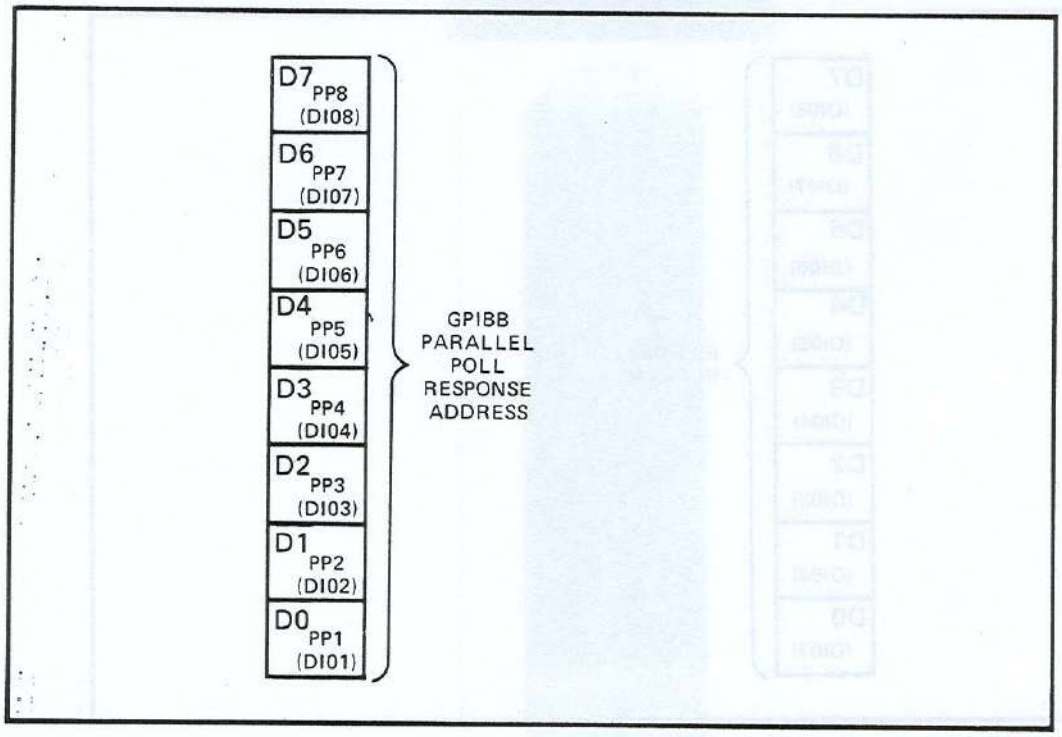
This register contains the byte supplied to the Controller In Charge when it is conducting a Serial Poll on the GPIBB. The register is cleared by a hardware Reset but not by a software **swrst** Auxiliary Command. Bits **S8** and **S6** through **S1** supply device dependent information to the Controller In Charge, while setting bit **S7** (**rsv**) forces bus line **SRQ** true. When the Controller responds by carrying out a Serial Poll on the GPIBB, the TMS 9914 controlled **SRQ** output automatically returns to the passive false (high) state. A new Service Request cannot be made until bit **rsv** is first reset, and then set again.

COMMAND PASS THROUGH REGISTER
IN 46h



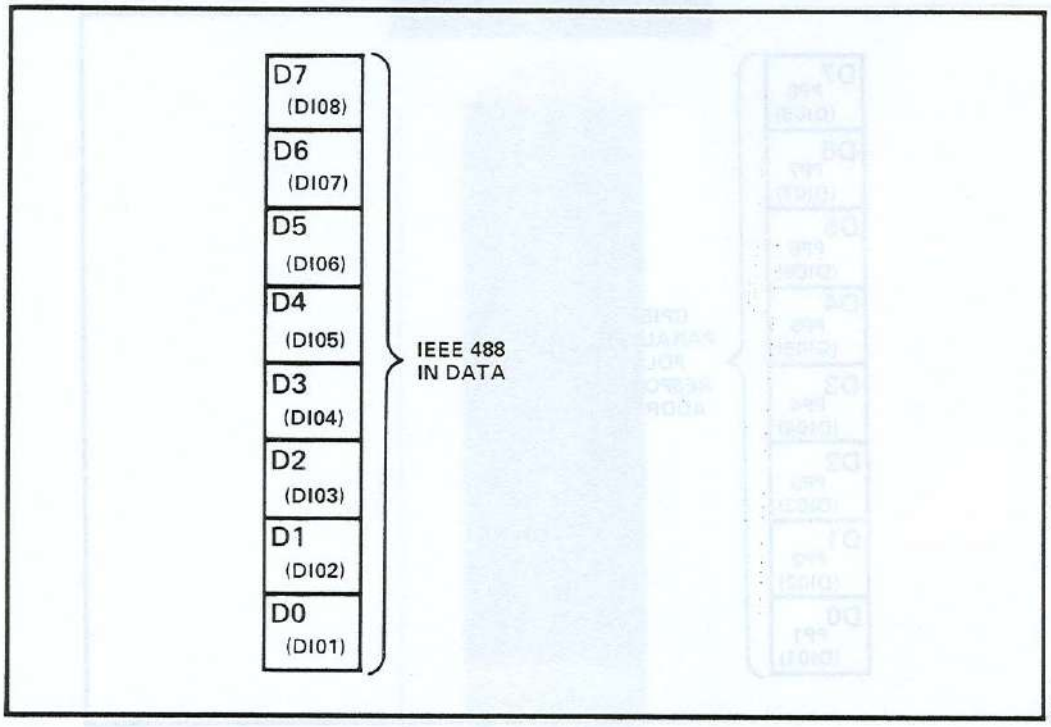
The GPIBB Z-80A reads this register to sample the IEEE 488 DIO8 - DIO1 data lines when an unrecognized command or a secondary address condition occurs. These bits are not latched, so the GPIBB Z-80A typically has to read this register in response to an interrupt generated by unmasked **GPIB Interrupt Mask 1** bits **UCG** and/or **APT.**

PARALLEL POLL REGISTER
OUT 46h



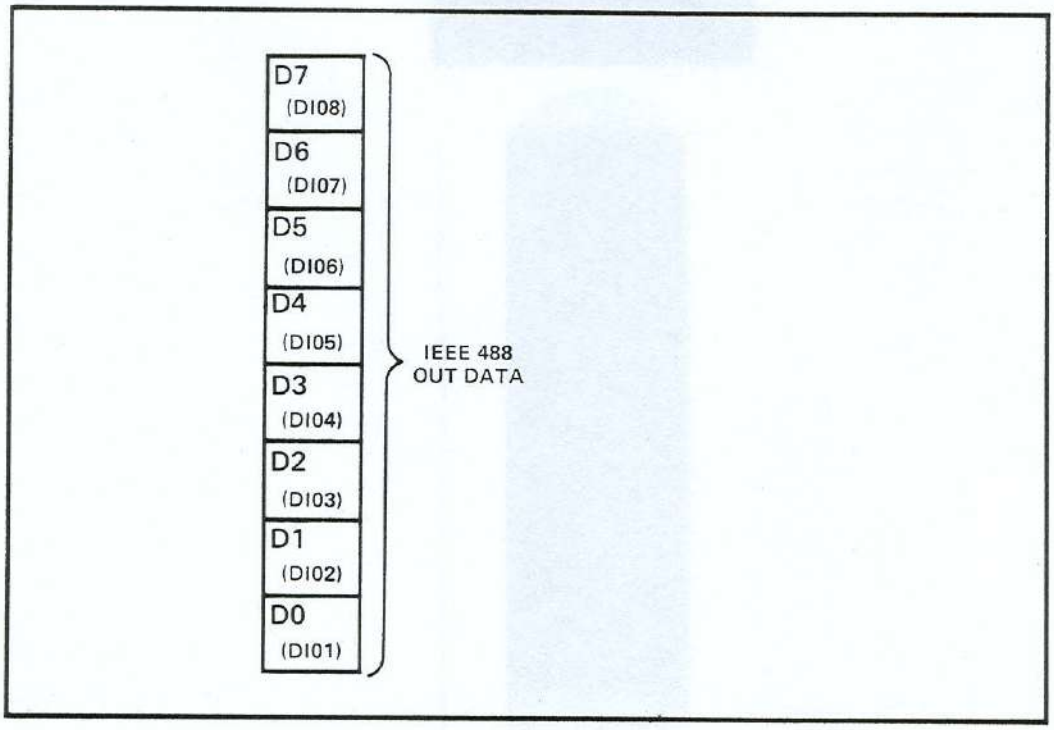
The contents of this register are placed on the IEEE 488 bus when the Controller In Charge conducts a Parallel Poll. The GPIBB Z-80A must load the register before the Parallel Poll operation occurs (usually during GPIBB initialization). The register may be remotely configured using the **pts** Auxiliary Command. This register is cleared by a TMS 9914 Reset, but not by a **swrst** Auxiliary Command.

DATA IN REGISTER
IN 47h



The GPIBB Z-80A reads an Input Byte from the IEEE 488 bus through this register when the TMS 9914 is addressed as a Listener. **GPIB Interrupt Status 0** bit **BI** remains set, and the Not Ready For Data line (**NRFD**) is held active low until this register is read. Reading this register also causes the TMS 9914 to releases bus line **NRFD** when reading this register unless either holdoff command **hdfa** or command **hdfe** has previously been issued. Issuing Auxiliary Command **rhdf** forces the **NRFD** handshake line inactive high, thereby completing the acceptor handshake.

DATA OUT REGISTER
OUT 47h



When the TMS 9914 is functioning in the Talker or Controller modes, this register is transfers data bytes or command bytes from the Z-80A to the IEEE 488 bus. If the TMS 9914 is in the Controller Active State, commands are sent with bus line **ATN** active low. If the TMS 9914 is in the Talker Active State, device dependent data is sent with bus line **ATN** inactive high. In both cases the source handshake function is automatically carried out by the TMS 9914. The GPIBB Z-80A may load this register with a new byte whenever **GPIB Interrupt Status 0** bit **BO** is set. Bit **BO** is set when the TMS 9914 enters the Talker mode to prompt initial loading.

Bus line **ATN** may at times be forced active low by the Controller In Charge after the **Data Out Register** is loaded but before the data appears on the IEEE 488 bus lines. This byte appears on the bus immediately after bus line **ATN** goes inactive high, unless Auxiliary Command **nbaf** is issued before the byte is placed on the bus.

Chapter 4

GPIBB INTERRUPT STRUCTURE

4.1 GPIBB TO HOST INTERRUPTS

The GPIBB can issue two types of interrupt requests to the host processor over the S-100 bus: non-maskable requests, which the host must acknowledge, and maskable interrupt requests, which the host may, or may not, acknowledge.

The GPIBB issues a non-maskable interrupt to the host by setting **GPIBB Control** bit **NMInt Host.** This causes S-100 bus line $\overline{\text{NMI}}$ to go active low on the next falling edge of S-100 bus signal sM1. This active low level is automatically removed on the following sM1 falling edge.

The GPIBB issues a maskable interrupt request to the host by setting **GPIBB Control** bit **Int Host.** The host processor may either:

1. Allow the GPIBB to drive S-100 bus line $\overline{\text{INT}}$ active low, or

2. Disallow this control, yet sense an interrupt request condition if polling the **GPIBB Control** bit **Int Host** indicates that the bit is set.

In the first case, the host processor sets **Host Control** bit **Int Mask** and resets bit **Clr Int.** S-100 bus line $\overline{\text{INT}}$ then goes active low when the GPIBB sets **GPIBB Control** bit **Int Host** (the GPIBB should have previously defined an interrupt vector value if the host Z-80 is operating in IM0 or IM2 -- see below). The host Z-80A then either:

- acknowledges the request if its own $\overline{\text{INT}}$ input pin is unmasked (interrupts enabled), or

- ignores the request if its $\overline{\text{INT}}$ pin is masked (interrupts disabled).

The GPIBB holds S-100 bus line $\overline{\text{INT}}$ active low until either:

- a host interrupt acknowledge cycle occurs, automatically clearing the request, or

- **Host Control** bit **Clr Int** is set removing the request and inhibiting further requests while the bit is set.

The GPIBB is required to supply an eight bit interrupt vector during interrupt acknowledge if the host is operating in Interrupt Mode 0 or Interrupt Mode 2. The GPIBB defines the interrupt vector value by writing a byte to port **GPIBB Output Data**; the contents of this port are gated onto the S-100 bus DI lines during host interrupt acknowledge.

In the second case, the host processor resets **Host Control** bits **Int Mask** and **Clr Int.** Then as the GPIBB sets **GPIBB Control** bit **Int Host**, S-100 bus line $\overline{\text{INT}}$ is unaffected, but the host can sense the interrupt request if polling reveals that the **Host Status** bit **Int Pend** is set. The host may then either:

- set control bit **Int Mask**, enabling S-100 bus line $\overline{\text{INT}}$ to be driven active low, or

- clear the interrupt request by setting bit **Clr Int** (removing the current interrupt request and inhibiting further interrupt requests), and then resetting bit **Clr Int** (allowing further interrupt requests to be sensed).

When two or more S-100 bus boards issue maskable interrupt requests to the host processor, their requests are coordinated by Cromemco's S-100 Interrupt Priority daisy chain. The S-100 interrupt daisy chain is controlled by $\overline{\text{PRIORITY IN}}$ and $\overline{\text{PRIORITY OUT}}$ lines provided on the following Cromemco products:

- the **GPIB Interface** at connector J3
- the **IOP** Input/Output Processor
- the **PRI** line printer interface
- the **TU-ART** serial interface
- the **4FDC** and **16FDC** floppy disk interfaces
- the **WDI** Winchester hard disk interface

The board priorities are defined as shown in Figure 4-1.

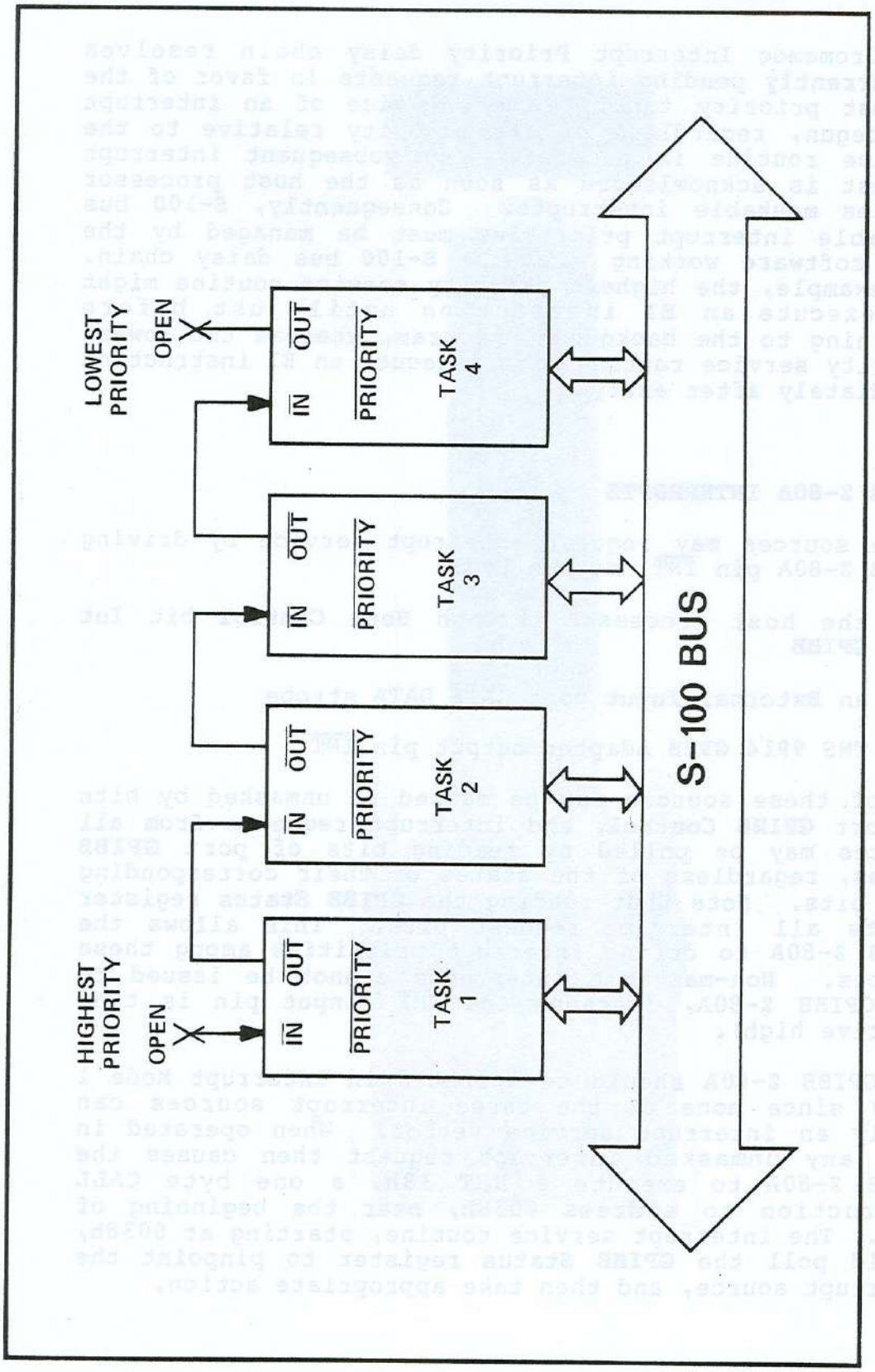


Figure 4-1: S-100 INTERRUPT DAISY CHAIN WIRING

The Cromemco Interrupt Priority daisy chain resolves **concurrently pending** interrupt requests in favor of the highest priority task. After service of an interrupt has begun, regardless of its priority relative to the service routine in progress, any subsequent interrupt request is acknowledged as soon as the host processor enables maskable interrupts. Consequently, S-100 bus maskable interrupt priorities must be managed by the host software working with the S-100 bus daisy chain. For example, the highest priority service routine might not execute an **EI** instruction until just before returning to the background program, whereas the lowest priority service routine might execute an **EI** instruction immediately after entry.

4.2 GPIBB Z-80A INTERRUPTS

Three sources may request interrupt service by driving GPIBB Z-80A pin $\overline{INT}$ active low:

1. the host processor through **Host Control** bit **Int GPIBB**
2. an **External Input** port GATE DATA strobe
3. TMS 9914 GPIB Adapter output pin $\overline{INT}$

All of these sources may be masked or unmasked by bits of port **GPIBB Control**, and interrupt requests from all sources may be polled by reading bits of port **GPIBB Status**, regardless of the states of their corresponding mask bits. Note that reading the **GPIBB Status** register resets all interrupt request bits. This allows the GPIBB Z-80A to define interrupt priorities among these sources. Non-maskable interrupts cannot be issued to the GPIBB Z-80A, (because its $\overline{NMI}$ input pin is tied inactive high).

The GPIBB Z-80A should be operated in Interrupt Mode 1 (IM1) since none of the three interrupt sources can supply an interrupt service vector. When operated in IM1, any unmasked interrupt request then causes the GPIBB Z-80A to execute a **RST 38h**, a one byte CALL instruction to address 0038h, near the beginning of ROM0. The interrupt service routine, starting at 0038h, should poll the **GPIBB Status** register to pinpoint the interrupt source, and then take appropriate action.

The host processor issues maskable interrupt requests to the GPIBB Z-80A by setting **Host Control** bit **GPIBB Int.** The request is acknowledged if **GPIBB Control** bit **Host Mask** is set and Z-80A maskable interrupts are enabled.

A maskable interrupt request is issued to the GPIBB Z-80A each time the **External Input** is loaded by pulsing GATE DATA (J1 pin 23) high. The request is acknowledged if **GPIBB Control** bit **Input Mask** is set and Z-80A maskable interrupts are enabled.

The TMS 9914 GPIB Adapter issues maskable interrupt requests to the GPIBB Z-80A by driving its $\overline{\text{INT}}$ output pin active low. The request is acknowledged if **GPIBB Control** bit **GPIB Mask** is set and Z-80A maskable interrupts are enabled. A TMS 9914 interrupt request is removed by reading port **GPIB Interrupt Status 0** and/or **GPIB Interrupt Status 1.**

Appendix A

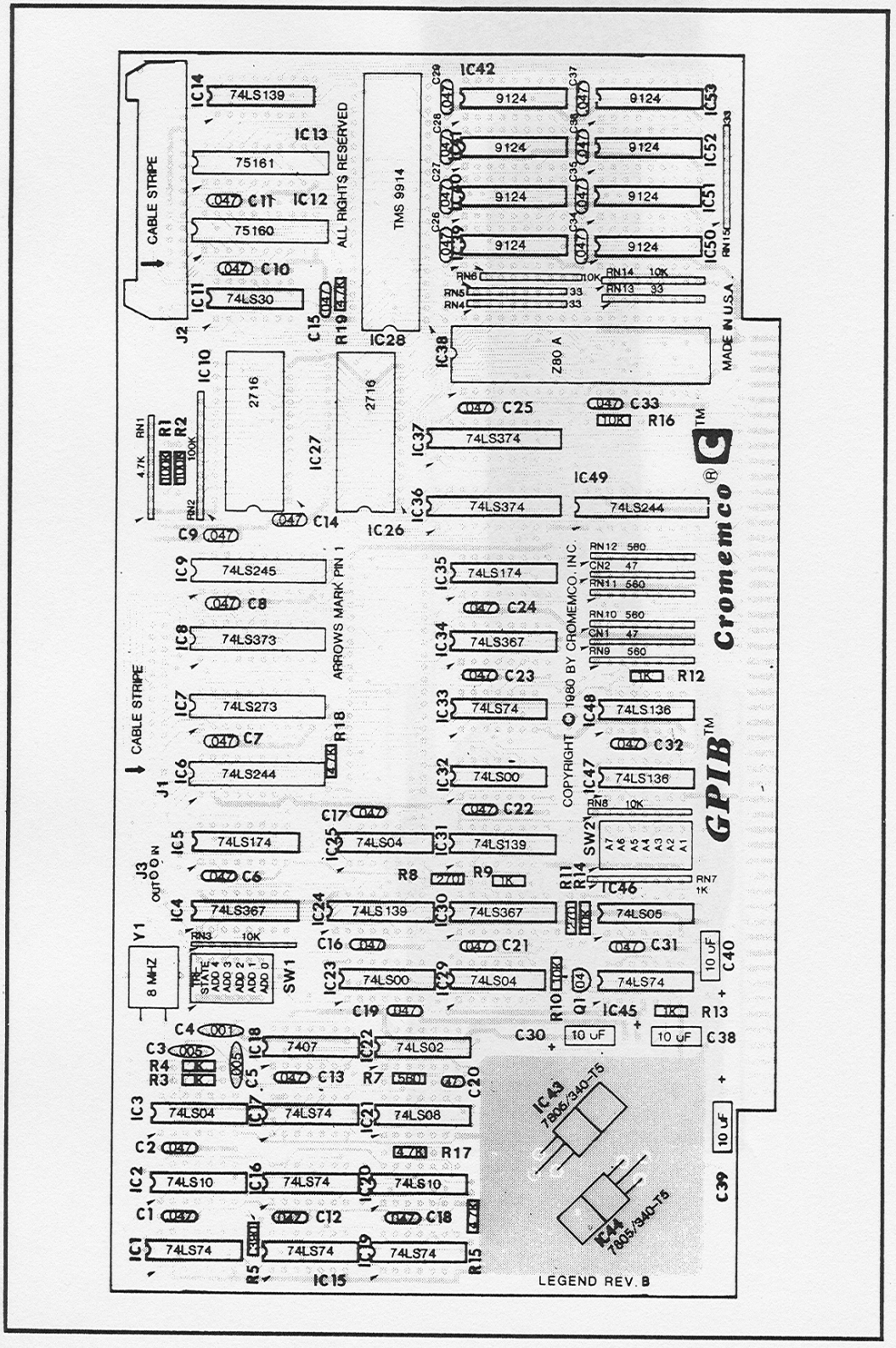
PARTS LIST

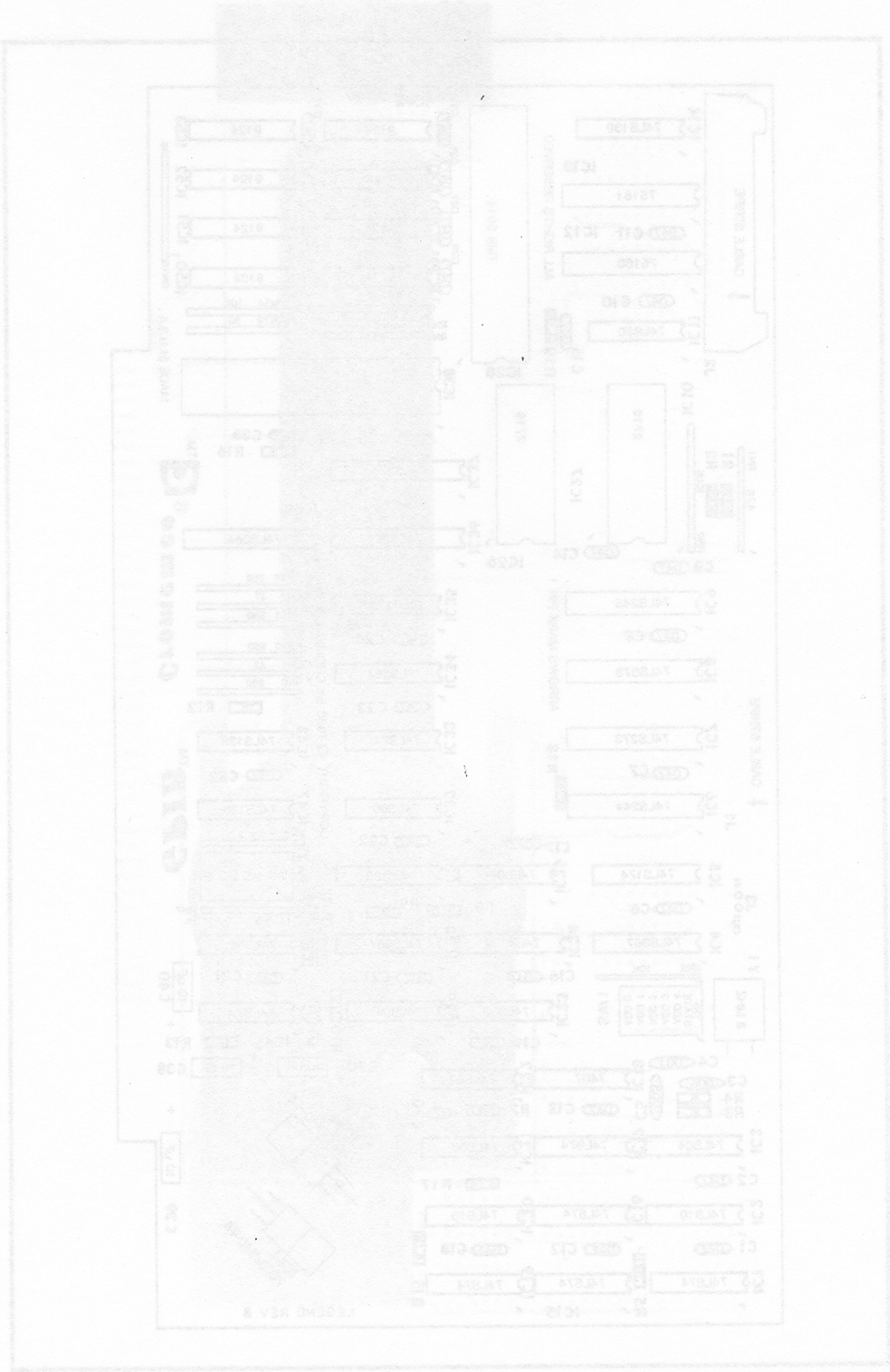
Designation	Description	Cromemco Part No.
Integrated Circuits		
IC1	74LS74	010-0055
IC2	74LS10	010-0063
IC3	(non TI) 74LS04	010-0066
IC4	74LS367	010-0108
IC5	74LS174	010-0097
IC6	74LS244	010-0100
IC7	74LS273	010-0107
IC8	74LS373	010-0102
IC9	74LS245	010-0120
IC10	GPIB Monitor ROM	502-0070
IC11	74LS30	010-0059
IC12	75160	010-0317
IC13	75161	010-0316
IC14	74LS139	010-0118
IC15-17	74LS74	010-0055
IC18	7407	010-0104
IC19	74LS74	010-0055
IC20	74LS10	010-0063
IC21	74LS08	010-0064
IC22	74LS02	010-0068
IC23	74LS00	010-0069
IC24	74LS139	010-0118
IC25	(non TI) 74LS04	010-0066
IC27	2716/TMS2516	not supplied
IC28	TMS 9914	011-0057
IC29	(non TI) 74LS04	010-0066
IC30	74LS367	010-0108
IC31	74LS139	010-0118
IC32	74LS00	010-0069
IC33	74LS74	010-0055
IC34	74LS367	010-0108
IC35	74LS174	010-0097
IC36,37	74LS374	010-0133
IC38	Z80-A	011-0010
IC39-42	AM9124EPC	011-0055
IC43,44	7805/340T-5	012-0001
IC45	74LS74	010-0055
IC46	74LS05	010-0065
IC47,48	74LS136	010-0050
IC49	74LS244	010-0100
IC50-53	AM9124EPC	011-0055

Designation	Description	Cromemco Part No.
Diodes		
Q1	2N3904	009-0001
Capacitors		
C1,2	.047 uf axial	004-0061
C3	.005 uf disk	004-0025
C4	.001 uf disk	004-0022
C5	.005 uf disk	004-0025
C6-19	.047 uf axial	004-0061
C20	47 pf mono	004-0000
C21-29	.047 uf axial	004-0061
C30	10 uf tant	004-0032
C31-37	.047 uf axial	004-0061
C38-40	10 uf tant	004-0032
Capacitor Networks		
CN1,2	47 PF, 8 pin	005-0000
Resistors		
R1,2	100 K	001-0039
R3,4	1 K	001-0018
R5	390	001-0013
R7	560	001-0015
R8	270	001-0011
R9	1 K	001-0018
R10	10 K	001-0030
R11	270	001-0011
R12,13	1 K	001-0018
R14	10 K	001-0030
R15	4.7 K	001-0024
R16	10 K	001-0030
R17-19	4.7 K	001-0024
Resistor Networks		
RN1	4.7 K, 8 pin	003-0030
RN2	100 K, 10 pin	003-0035
RN3	10 K, 8 pin	003-0025
RN4,5	33, 8 pin	003-0000
RN6	10 K, 8 pin	003-0025

Designation	Description	Cromemco Part No.
RN7	1 K, 1 pin	003-0007
RN8	10 K, 8 pin	003-0025
RN9-12	560, 8 pin	003-0006
RN13	33, 8 pin	003-0000
RN14	10 K, 8 pin	003-0025
RN15	33, 8 pin	003-0000
Miscellaneous		
	1 switch, 6 pos.	013-0025
	1 switch, 7 pos.	013-0033
	4 6-32 hex nut	015-0013
	4 #6 lock washer	015-0020
	4 6-32X1/2 screw	015-0044
	8 socket, 16 pin	017-0002
	8 socket, 18 pin	017-0003
	9 socket, 20 pin	017-0004
	2 socket, 24 pin	017-0005
	2 socket, 40 pin	017-0006
	1 socket, 2 pin	017-0009
	1 large heat sink	021-0017
	1 crystal, 8 mhz	026-0001

Appendix B


CABLE STRIPE
ALL RIGHTS RESERVED
TMS 9914
Z80 A
MADE IN U.S.A.
2716
2716
ARROWS MARK PIN 1
COPYRIGHT © 1980 BY CROMEMCO, INC.
Cromemco®
GPIB™
8 MHZ
SW1
SW2
LEGEND REV. B



Appendix C

GPIB MONITOR COMMANDS

The GPIB Monitor is controlled by single and double character commands that may be followed by one or more arguments. This section describes these commands and gives the command format for each.

All GPIB Monitor commands must be terminated with a RETURN character.

@ -- EXECUTE BATCH COMMAND STRING

The @ command executes a command string located in GPIB memory.

Format:

@ address

The ASCII string, beginning at **address** in GPIB memory, and ending with either a semicolon or a binary 0, is taken as a list of commands to be executed by GPIBMON. The string may have multiple commands within it. An error message is displayed and control is returned to the GPIBMON command level if an error occurs during command processing.

DM -- DISPLAY MEMORY

The **DM** command formats and displays the contents of GPIB memory on the console screen.

Formats:

dm
dm start-address
dm start-address end-address
dm start-address S swath
dm S swath

The letter **m** is optional in all formats.

The contents of memory are displayed on the console in both hexadecimal and ASCII. Each display line provides information on up to 16 consecutive bytes of memory. Each line consists of three information fields. The leftmost field is the memory address, displayed in hexadecimal, of the first byte displayed in the center field. The center field contains the hexadecimal values of from 1 to 16 consecutive data bytes. The rightmost field displays the ASCII equivalent characters of each displayed byte. Since the 7-bit ASCII code ignores MSB bit D7, then ASCII character **A** is displayed for both data byte 41h and C1h. If the data byte is not equivalent to a printable ASCII character, the rightmost field will show a period character.

Example Display:

```
0100 41 42 43 44 30 31 32 33 C3 02 55 00 0D 0A 24 FF ABCD0123C.U...$.
```

The first command format above displays 80h bytes starting with the ending address of the previous **DM** command, plus 1 (or 4000h if a **DM** command has not previously been issued). The second format displays 80h consecutive bytes beginning at **start-address.** The third displays all bytes between **start-address** and **end-address.** The fourth displays **swath** bytes starting with **start-address.** The last form displays **swath** bytes, starting with the ending address of the previous **DM** command, plus 1 (or 4000h if a **DM** command has not previously been issued). A default value of 80h is assumed in the last two forms if no **swath** value follows swath character **S.**

E -- EXAMINE INPUT PORT

The **E** command reads data from input port number **port-address,** and displays the data in hexadecimal on the console screen.

Format:

e port-address

G -- GO TO ABSOLUTE ADDRESS

The **G** command jumps to **address** in GPIB memory with current GP Z-80A register values.

Format:

g address

This command transfers program control from GPIBMON to the program starting at **address** in GPIB memory.

M -- MOVE (COPY) A BLOCK OF MEMORY

The **M** command copies the contents from one block of GPIB memory to another block of GPIB RAM memory. The contents of the original block are unchanged unless the two memory blocks overlap.

Formats:

m source-address end-address destination-address
m source-address S swath destination-address
m S swath destination

The first command format copies all bytes from the **source-address** byte through the **end-address** byte to **destination-address.** The second copies **swath** bytes from **source-address** to **destination-address.** The last copies **swath** bytes from 4000h to **destination-address.**

After the move, GPIBMON verifies that the two blocks of memory are the same. Discrepancies are formatted on the console as follows:

ssss xx yy dddd

where **ssss** is the source address, **xx** is the source data, **yy** is the destination data, and **dddd** is the destination address. The verification process lists discrepancies which are not necessarily errors after certain types of overlapping moves. Displays of this type may be terminated by typing **CONTROL-C, CONTROL-Z** or **ESCAPE.**

O -- OUTPUT DATA TO A PORT

The **O** command writes **data-byte** to output port number **port-address.**

Format:

o data-byte port-address

Q -- QUERY MEMORY FOR A STRING OF BYTES

The **Q** command searches GPIBMON memory for a string.

Formats:

q start-address end-address string-of-bytes
q start-address S swath string-of-bytes

This command searches memory **start-address** through **end-address,** or **swath** bytes starting at **start-address,** for **string-of-bytes. String-of-bytes** can have any of the formats allowed by the **SM** command. If **string-of-bytes** is found, then 16 bytes are displayed, starting with the first matching byte.

R -- READ BINARY DATA FROM THE CONSOLE

The **R** command reads unaltered bytes from the current console and writes them to GPIB RAM memory.

Formats:

r start-address end-address
r start-address S swath

This command reads bytes (all 8 bits) from the current console and sequentially writes them into memory **start-address** through **end-address, or swath** bytes starting at **start-address,** until all bytes are written.

This command may be used for loading binary or ASCII data from a Teletype paper tape reader into GPIB RAM memory.

SM -- SUBSTITUTE MEMORY

The **SM** command changes the contents of GPIB RAM memory.

Formats:

sm
sm address

The character **m** is optional in both formats.

The first command format prompts the user to change memory at the last location changed, plus 1 (4000h if no **SM** command has previously been issued). The second command form prompts the user to change memory at **address.** In both cases, GPIBMON prompts the user by displaying **memory-address** followed by its **current-contents.** The user has six options:

1. Type a **data-byte** value followed by **RETURN.** This causes **data-byte** to be stored at **memory-address. Memory-address** is then incremented and displayed on the next console line.

2. Type a **'string'** value followed by **RETURN.** This causes the ASCII code for all **string** characters (between the apostrophes) to be sequentially stored starting at **memory-address. Memory-address** is then adjusted beyond the last **string** byte, and is displayed on the next console line.

3. Any combination of **data-bytes** and **'strings'** may be entered in the same line, separated by SPACEs, with one **RETURN** terminating the line. **Memory-address** is then adjusted beyond the last byte stored, and displayed on the next console line.

4. Type a minus sign. This causes **memory-address** to decrement with no memory change. The new **memory-address** is displayed on the next console line. This option is used to "back up" and correct a previous **memory-address.**

5. Type a **RETURN.** This causes **memory-address** to increment with no memory change, and the new **memory-address** to be displayed on the next console line.

6. Type a period. This terminates the **SM** dialogue, and GPIBMON returns to the GPIBMON command level.

V -- VERIFY A BLOCK OF MEMORY

The **V** command verifies that the contents of two blocks of memory are the same.

Formats:

v source-address end-address destination-address
v source-address S swath destination-address

The first command format verifies that all bytes from **source-address** through **end-address** match the same number of bytes starting at **destination-address.** The second verifies that **swath** bytes beginning at **source-address** match **swath** bytes starting at **destination-address.**

Discrepancies are formatted as follows:

ssss xx yy dddd

where **ssss** is the source address, **xx** is the source data, **yy** is the destination data, and **dddd** is the destination address.

W -- WRITE BINARY DATA FROM MEMORY TO THE CONSOLE

The **W** command sequentially writes a block of GPIB memory bytes to the console.

Formats:

w start-address end-address
w start-address S swath

This command sequentially reads bytes from memory **start-address** through **end-address,** or **swath** bytes starting at **start-address.** The command writes them (all 8 bits) to the current console until all bytes are written. This command may be used for writing GPIB memory to a teletype paper tape punch.

Z -- ZAP MEMORY WITH A BYTE CONSTANT

The **Z** command fills a block of GPIB RAM memory with the same single byte value.

Formats:

z start-address end-address [value]
z start-address S swath [value]

This command fills GPIB RAM memory **start-address** through **end-address**, or **swath** bytes starting at **start-address**, with byte **value.** **Value** may either be a numeric or character value, e.g., **100.**, or **'?'.** If no **value** is specified, 00h is assumed.

Appendix D

SAMPLE GPIB PROGRAMS

```
        TITLE   GPIB    GPIB Board Monitor
        SUBTTL  *** Initialization and Main Control ***
        NAME    GPIB

;;      Version Definitions

VERNUM  EQU     01                      ; Version #
RLSNUM  EQU     00                      ; Release #

;;      The monitor starts off from here
;;      Skip around the user interface portion

START:  JR      START1                  ; Skip around user info & RST 1

;;      The following 6 bytes are for
;;      user program reference

        DB      VERNUM,RLSNUM           ; Version # for user reference
        DW      CONNUM                  ; Pointer to console data
        DW      XXXXXX                  ; Free space pointer

;;      When a JSYS is executed, program control
;;      transfers to here for further processing

        ORG     START+08H               ; ORG at RST 1
        JP      JSYSX                   ; User entry control transfer point

;;      Initialize the monitor RAM and the console, print the
;;      signon message, then continue after the RST locations

START1: LD      SP,STACK                ; Load the SP
        LD      HL,VARBS                ; Clear the variables
        LD      A,STACK-VARBS-4         ; in high memory
        CALL    ZERO                    ; Zap! All clean
        CALL    TTYIO
        CALL    OUTSTI                  ; Print signon message
        DB      'GPIB Monitor '
VERTXT: DB      VERNUM/10+'0',VERNUM%10+'0','.'
        DM      RLSNUM/10+'0',RLSNUM%10+'0'
        JR      START2                  ; Skip around RST 6 & RST 7

;;      If a break instruction (RST 30H) is executed, control will
;;      transfer to here, then back to DEBUG, provided it is running

        ORG     START+30H               ; DEBUG break restart transfer
        JP      BREAK                   ; in high memory

;;      More user interface information
;;      (otherwise unuseable due to proximity)

        DW      SETMM1                  ; Pointer to SM command (for DEBUG)
        DW      -1                      ; Pointer to INIT single console
        DB      -1                      ; Reserved for future use?
        FORM

;;      If an invalid jump occurs, chances are that the PC will
;;      reach and execute an RST 38H instruction: the following
;;      will intercept it and display an error message with
;;      the bad address, then return control to the monitor

        ORG     START+38H               ; Crash trap location
CRASH:  POP     HL                      ; Get crash PC
        LD      SP,STACK                ; Reload SP
        CALL    OUTSTI                  ; Print error message
        DM      CR,BEL,'Crash '         ; about what happened
        DEC     HL                      ; Adjust the PC
        CALL    HEXWO                   ; Print it

;;      Go to the next line after a message,
;;      then initialize hi-segment variables

START2: CALL    CRLF                    ; Go to next line
        LD      HL,4000H                ; Set initial pointers to 4000H
        LD      (SETPNT),HL             ; for Substitute Memory command
        LD      (DISPNT),HL             ; and Display Memory command
        LD      A,0C3H                  ; Set up JP instruction
        LD      (BREAK),A               ; at BREAK to abort
START3: LD      HL,CRASH                ; the job incase DEBUG
        LD      (BREAK+1),HL            ; isn't in the system

;;      Main command level:
;;      If not a BATCH job, then prompt for a
;;      command, and get a command from user
;;      If a BATCH job, point to the next command and proceed

REENTE: LD      SP,STACK                ; Reload SP
        LD      A,(BATERR)              ; Get BATCH error flag
        OR      A                       ; Are any errors?
        JR      NZ,REENTX               ; Yes, reset & abort BATCH job
        LD      A,(BATFLG)              ; Get BATCH job flag
        LD      DE,(BATPNT)             ; Get BATCH pointer in case active
        OR      A                       ; Is there a BATCH job active?
        JR      NZ,REENT1               ; Yes, continue w/o prompt
        CALL    OUTSTI                  ; Print the prompt
        DM      '2.'                    ; (Nice & simple)
        LD      DE,LINBUF               ; Point to input buffer
        LD      A,LINLEN                ; Get line length
        CALL    INLINE                  ; Get input line
```

```
REENT1: CALL    NTSSCN                  ; Check first non-space
        OR      A                       ; Is it end of line?
        JR      Z,REENTX                ; Yes, ignore & reset BATCH flags
        CP      ';'                     ; Remainder of line a comment?
        JR      Z,REENTX                ; Yes, ignore & reset BATCH flags
        CALL    LCUC                    ; Convert command to uppercase
        CP      'Z'+1                   ; Too big for "Zap" command?
        JR      NC,CMDERR               ; Yes, error
        SUB     '@'                     ; Doodle off ASCII bias
        JR      C,CMDERR                ; Oops, not a command type
        ADD     A                       ; *2 for command jump table
        LD      HL,CMDJMP               ; Point to command jump table
        CALL    ADDH                    ; Point to command entry
        CALL    LOADHH                  ; Get the command routine address
        INC     DE                      ; Bump past command letter
        CALL    NTSSCN                  ; Get next character
        CALL    LCUC                    ; Convert to uppercase if needed
        LD      BC,REENTE               ; Point to re-entry location
        PUSH    BC                      ; Save it for return trip
        PUSH    HL                      ; Save the routine address
        LD      HL,4000H                ; Load default argl if needed
        RET                             ; Go to the routine

;;      Come here for absolute job abort; nothing at this point
;;      can keep the monitor from going back to the main prompt

REENTC: XOR     A                       ; Say no more BREAK in HISEG
        LD      (BREAK+2),A             ; Save it now that DEBUG's gone
        JR      REENTX                  ; Cancel whatever else exists

;;      Come here if command (or other) error

CMDERR: CALL    OUTSTI                  ; Print error message
        DM      BEL,'?\R'               ; "<BEL>?<CR><LF>"

;;      This part will return to the routine pointed to by XCTADR ONLY
;;      if the BREAK jump transfer is not pointing to the monitor, else
;;      go back to main command level (no BATCH jobs will survive)
;;      If the console is not at the beginning of an
;;      output line then print a <CR><LF> sequence

REENTX: LD      HL,0                    ; Reset BATCH flag and
        LD      (BATDAT),HL             ; say no BATCH errors
REENTQ: LD      A,(TTYPHP)              ; Get the current console position
        OR      A                       ; At beginning of a line?
        CALL    NZ,CRLF                 ; No, go there
        LD      A,(BREAK+2)             ; Get BREAK's JP address
        OR      A                       ; Is DEBUG running?
        JR      Z,START3                ; No, go back to command level
        LD      HL,(XCTADR)             ; Yes, get execution return address
        JP      (HL)                    ; Go back to wherever


        SUBTTL  *** Move & Verify Commands ***

;;      Move Memory Command
;;      M source S swath dest <CR>
;;      M source finish dest <CR>
;;      This command moves swath bytes from source to destination
;;      (or from source through finish to destination)

MOVE:   CALL    ARG3Q                   ; Get arguments
        PUSH    BC                      ; Save swath
        PUSH    DE                      ; and destination
        PUSH    HL                      ; and source
        LDIR                            ; Boing! New location
        POP     HL                      ; Get back arguments
        POP     DE                      ; for verify routine
        POP     BC                      ; Need length, too
        JR      VERIFX                  ; Go verify

;;      Verify Memory Command
;;      V source S swath dest <CR>
;;      V source finish dest <CR>
;;      This command compares (verifies) swath bytes from
;;      source to destination (or from source through finish
;;      to destination) and displays any discrepancies

VERIFY: CALL    ARG3Q                   ; Get the arguments

VERIFX: LD      A,(DE)                  ; Get data @ destination
        CP      (HL)                    ; Is it the same as @ source?
        JR      Z,VERIFO                ; Yes, check next byte
        CALL    HEXWO                   ; No, print source address
        LD      A,(HL)                  ; Get source data
        CALL    HEXBO                   ; Print it
        CALL    SPACE                   ; Move over
        LD      A,(DE)                  ; Get destination data
        CALL    HEXBO                   ; Print it
        CALL    SPACE                   ; Gap
        EX      DE,HL                   ; Get destination address
        CALL    HEXWO                   ; Print it
        EX      DE,HL                   ; Restore it
        CALL    CRLF                    ; New line
```

```
VERIFO: INC     DE                      ; Bump destination pointer
        INC     HL                      ; Bump source pointer
        DEC     BC                      ; Drop count
        LD      A,B                     ; Did it go
        OR      C                       ; to 0?
        JR      NZ,VERIFX               ; No, keep checking
        RET                             ; Yes, return

        SUBTTL  *** Set Memory Command ***

;;      Set (Substitute) Memory Command
;;      SM addr <CR>
;;      This command will prompt for information to
;;      be stuffed into memory (this command kills
;;      BATCH jobs due to possible buffer conflicts)
;;      The letter M is optional

SETMEM: CP      'M'                     ; Is "S" followed by "M"?
        JR      NZ,SETMM0               ; No, just "S"
        INC     DE                      ; Bump past "M"
SETMM0: LD      HL,(SETPNT)             ; Get default argument
        CALL    ARG1D                   ; Get address where to start

SETMM1: XOR     A                       ; Get a 0
        LD      (BATFLG),A              ; Abort any current BATCH job
        CALL    HEXWO                   ; Print address
        LD      A,(HL)                  ; Get current contents
        CALL    HEXBO                   ; Print them
        CALL    SPACE                   ; Print a space

        LD      DE,LINBUF               ; Point to line buffer
        LD      A,LINLEN                ; Get line length
        CALL    INLINE                  ; Get input line
        CALL    NTSSCN                  ; Scan for non-space
        CP      '.'                     ; Is it to stop?
        RET     Z                       ; Yes, return
        CP      '-'                     ; Is it to back up?
        JR      NZ,SETMM3               ; No, do the conversion
SETMM2: DEC     HL                      ; Back pointer up
        JR      SETMM6                  ; Go save it & try again

SETMM3: CALL    INSTR                   ; Convert input line to binary
        JR      NC,SETMM4               ; If no error, then continue
        CALL    OUTSTI                  ; Print error message
        DM      BEL,'?',CR              ; "<BEL>?<CRLF>"
        JR      SETMM1                  ; & try again

SETMM4: XOR     A                       ; Zap A
        ADD     B                       ; to check count in B
        JR      NZ,SETMM5               ; Not empty, save the data
        INC     HL                      ; Bump pointer
        JR      SETMM6                  ; & try again

SETMM5: LD      C,B                     ; Load count in BC
        LD      B,0                     ; proper
        EX      DE,HL                   ; Swap pointers
        LDIR                            ; Shove the data into memory
        EX      DE,HL                   ; Restore pointers

SETMM6: LD      (SETPNT),HL             ; Save pointer for later
        JR      SETMM1                  ; & go for more

        SUBTTL  *** Find a String-of-Bytes Command ***

;;      Find a String-of-Bytes Command, also known as "Query"
;;      Q start S swath string-of-bytes <CR>
;;      Q start finish string-of-bytes <CR>
;;      This command searches the memory specified from
;;      start for swath bytes (or from start through
;;      finish) for the specified string-of-bytes;
;;      when found, the first 16 bytes are displayed

FIND:   CALL    ARG2                    ; Get address and Swath arguments
        JR      C,FINDE                 ; If none, then error
        PUSH    BC                      ; Save Swath
        CALL    INSTR                   ; Get string
        POP     DE                      ; Get Swath back into DE
FINDE:  JP      C,CMDERR                ; If error, then abort
        XOR     A                       ; Zap A
        ADD     B                       ; to check length of entry
        JP      Z,CMDERR                ; Oops! No string there

FIND1:  PUSH    BC                      ; Save length
        PUSH    DE                      ; and Swath
        PUSH    HL                      ; and address
        LD      DE,LINBUF               ; Point to binary buffer

FIND2:  LD      A,(DE)                  ; Get a byte
        CP      (HL)                    ; Is it the same?
        JR      NZ,FIND3                ; No, failure here
        INC     DE                      ; Bump binary pointer
        INC     HL                      ; Bump memory pointer
        DJNZ    FIND2                   ; Go until all expended

;;      When we get to here, the strings are the same
```

```
FIND3:  POP     HL                      ; Get memory pointer back
        PUSH    HL                      ; momentarily
        LD      B,10H                   ; Print 10 bytes
        CALL    Z,DISLIN                ; only if the strings matched
        POP     HL                      ; Get memory pointer back again
        POP     DE                      ; and Swath
        POP     BC                      ; and count

        INC     HL                      ; Bump memory pointer
        DEC     DE                      ; Drop Swath
        LD      A,D                     ; Did the Swath
        OR      E                       ; go to 0?
        JR      NZ,FIND1                ; No, keep looking
        RET                             ; Yes, all done here

        SUBTTL  *** String Input Processing Routine ***

;;      Get String of Bytes from (DE) & leave final
;;      string at DE in LINBUF with count in reg B
;;      Entry:  DE points to string
;;      Exit:   B contains count of bytes
;;              DE points to string in LINBUF

INSTR:  PUSH    HL                      ; Save HL
        LD      B,0                     ; Initial count of 0
        LD      HL,LINBUF               ; Point to binary result buffer
INSTR1: CALL    NCMSCN                  ; Skip spaces
        LD      (BATPNT),DE             ; Save BATCH pointer for later
        OR      A                       ; Is it EOL?
        JR      Z,INSTRD                ; Yes, done
        CP      ';'                     ; Rest of line a comment?
        JR      Z,INSTRD                ; Yes, done
        INC     DE                      ; Bump past current character
        CP      MXCHR                   ; Is it multi-command?
        JR      NZ,INSTRC               ; No, continue
        LD      (BATPNT),DE             ; Yes, save BATCH pointer
        LD      (BATFLG),A              ; Say BATCH active
        JR      INSTRD                  ; Finish up here
INSTRC: LD      C,A                     ; Save character in case delimiter
        CP      ''''                    ; Is it single quote?
        JR      Z,INSTRS                ; Yes, process string
        CP      '"'                     ; Is it double quote?
        JR      Z,INSTRS                ; Yes, process string
        DEC     DE                      ; Repoint to numeric arg
        PUSH    HL                      ; Save binary pointer
        CALL    ARGH                    ; Get argument
        LD      A,L                     ; Get result into A
        POP     HL                      ; Restore binary pointer
        JR      C,INSTRD                ; If no argument, then error
        LD      (HL),A                  ; Save argument
        INC     HL                      ; Bump binary pointer
        INC     B                       ; Bump count
        JR      INSTR1                  ; & go for more data

INSTRS: LD      A,(DE)                  ; Get the character
        INC     DE                      ; Bump past it
        OR      A                       ; Is it EOL?
        JR      Z,INSTRD                ; Yes, done
        CP      C                       ; Is it the delimiter?
        JR      Z,INSTR1                ; Yes, end of string
        LD      (HL),A                  ; Save the character
        INC     HL                      ; Bump binary pointer
        INC     B                       ; Bump count
        JR      INSTRS                  ; Go for next character

INSTRD: LD      DE,LINBUF               ; Point to beginning of buffer
        POP     HL                      ; Restore HL
        RET                             ; & return

        SUBTTL  *** Display Memory Command ***

;;      Display Memory Command
;;      DM start S swath <CR>
;;      DM start finish <CR>
;;      This command displays the contents of memory beginning
;;      at start for swath bytes (or from start through finish)
;;      If the start is missing, last DM address is assumed
;;      If the swath is missing, 80H is assumed
;;      The letter M is optional

DISPLY: CP      'M'                     ; Is "D" followed by "M"?
        JR      NZ,DISPL1               ; No, just "D"
        INC     DE                      ; Bump past "M"
DISPL1: LD      BC,80H                  ; Default swath is 80H bytes
        LD      HL,(DISPNT)             ; Get default data pointer
        CALL    ARG2D                   ; Get (new) arguments
DISMM1: LD      E,10H                   ; Assume line length of 10H
        XOR     A                       ; Zap A to
        OR      B                       ; check if near the end
        JR      NZ,DISMM2               ; No, continue
        LD      A,0FH                   ; Is there less than 10H
        CP      C                       ; bytes to go?
        JR      C,DISMM2                ; No, still assume 10H
        XOR     A                       ; Zap A to check
        OR      C                       ; if Swath 0?
        JR      Z,DISMM2                ; Yes, assume 10H to dump all
        LD      E,C                     ; Get final amount
```

```
DISMM2: PUSH    BC                      ; Save count to go
        LD      B,E                     ; Load line count
        CALL    DISLIN                  ; Print the line
        LD      (DISPNT),HL             ; Save data address
        POP     BC                      ; Get count back
        LD      A,C                     ; Adjust the count
        SUB     E                       ; by subtracting
        LD      C,A                     ; the number
        JR      NC,DISMM3               ; of bytes
        DEC     B                       ; just printed
DISMM3: LD      A,B                     ; Did we
        OR      C                       ; run out?
        JR      NZ,DISMM1               ; No, not done yet
        RET                             ; Yes, return

        SUBTTL  *** Display a line of memory up to 16 bytes ***

;;      Display up to 16 bytes of memory on the console
;;      Entry:  B contains the length
;;              HL points to the data

DISLIN: CALL    HEXWO                   ; Print address
        PUSH    BC                      ; Save count
        PUSH    HL                      ; Save data pointer
        LD      C,0                     ; 0 bytes printed
DISLN1: LD      A,11B                   ; Check if multiple
        AND     C                       ; of 4 bytes printed
        CALL    Z,SPACE                 ; Yes, print a space
        LD      A,(HL)                  ; Get data byte
        CALL    HEXBO                   ; Print it
        INC     HL                      ; Bump data pointer
        INC     C                       ; Bump byte count
        DJNZ    DISLN1                  ; Go until all has been dumped
DISLN2: LD      B,58                    ; Move out to column 58
        CALL    PRITAB                  ; Go there
        POP     HL                      ; Get data pointer back
        POP     BC                      ; & count, too
DISLN3: LD      A,(HL)                  ; Get character
        INC     HL                      ; Bump pointer
        CALL    DISLN4                  ; Print the character
        DJNZ    DISLN3                  ; & go until this line done
        JP      CRLF                    ; then goto next line & return

DISLN4: AND     ^^7                     ; Mask off ^7
        CP      DEL                     ; Is it <DEL>?
        JR      Z,DISLN5                ; Yes, non-graphic print
        CP      ' '                     ; Is it graphic?
        JP      NC,OUTCHR               ; Yes, go print it
DISLN5: LD      A,'.'                   ; Get a '.' instead of whatever
        JP      OUTCHR                  ; & go print it

        SUBTTL  *** Miscellaneous Other Commands ***

;;      Execute command string in memory at addr
;;      @ addr <CR>
;;      This command starts up a BATCH job as
;;      a sequence of commands to be executed

BATCH:  CALL    ARG1Q                   ; Get the string address
BATCH1: LD      (BATPNT),HL             ; Save it for BATCH processing
        LD      A,-1                    ; Say that BATCH
        LD      (BATFLG),A              ; is active
        RET                             ; Return

;;      Examine input port
;;      E port <CR>
;;      The contents of the input port are displayed on the console

EXAMIN: CALL    ARG1Q                   ; Get argument where to look
        LD      C,L                     ; Load into C
        IN      A,(C)                   ; & get the data
        CALL    HEXOUT                  ; Print it
        JP      CRLF                    ; New line & return

;;      Output data to a port
;;      O data port <CR>
;;      The data specified is sent to the port specified

OUTPUT: CALL    ARGH                    ; Get data byte
        PUSH    HL                      ; Save it on the stack
        CALL    ARG1Q                   ; Get port #
        LD      C,L                     ; Load into C
        POP     HL                      ; Get data byte back
        OUT     (C),L                   ; Send it
        RET                             ; & return

;;      Zap memory with a byte constant
;;      Z start S swath byte <CR>
;;      Z start finish byte <CR>
;;      This command Zaps the memory specified from
;;      start for swath bytes (or from start through
;;      finish) with the specified byte
```

```
ZAP:    CALL    ARG3Q                   ; Get addresses and data byte
        LD      (HL),E                  ; Save the data byte in memory
        LD      D,H                     ; Copy start address
        LD      E,L                     ; from HL to DE
        DEC     BC                      ; Adjust Swath for first byte
        LD      A,B                     ; Was "S1"
        OR      C                       ; specified?
        RET     Z                       ; Yes, already done
        INC     DE                      ; Bump destination to next byte
        LDIR                            ; ZAP!
        RET                             ; All done, return
        FORM

;;      Program Control Transfer; "Go" command
;;      G addr <CR>
;;      This command jumps to location addr

GOTO:   CALL    ARG1Q                   ; Get the address
        JP      (HL)                    ; Go to HL

;;      Send Nulls to the Console
;;      N <CR>
;;      N number <CR>
;;      This command issues number of <NUL> characters
;;      to the current console (default number is 16)

NULLS:  LD      HL,16                   ; Default # is 16 nulls
        CALL    ARG1D                   ; Get the number
        LD      B,L                     ; Save count in B
NULLS1: XOR     A                       ; Get a <NUL>
        CALL    TTYOUT                  ; Send the null
        DJNZ    NULLS1                  ; Go until all sent
        RET                             ; then return

;;      Read Binary Data from the console into memory
;;      R start S swath <CR>
;;      R start finish <CR>
;;      This command reads binary data from the current
;;       console into the IOP memory as specified

READ:   CALL    ARG2Q                   ; Get start & swath
READ1:  CALL    TTYIN                   ; Get a byte
        LD      (HL),A                  ; Save it in memory
        CPI                             ; Bump pointer & check count
        RET     NV                      ; Done, return
        JR      READ1                   ; Keep reading

;;      Write Binary Data from memory to the console
;;      W start S swath <CR>
;;      W start finish <CR>
;;      This command writes binary data from the IOP
;;       memory specified to the current console

WRITE:  CALL    ARG2Q                   ; Get start & swath
WRITE1: LD      A,(HL)                  ; Get a byte from memory
        CALL    TTYOUT                  ; Send it
        CPI                             ; Bump pointer & check count
        RET     NV                      ; Done, return
        JR      WRITE1                  ; Keep writing


        SUBTTL  *** User Operation Processing ***

;;      This routine dispatches user operation requests.  A "JSYS n"
;;      operation is executed by the user, where n is the function
;;      code.  Control is transferred here and the function code is
;;      converted to a routine address.  The PC is adjusted around
;;      the function code, then the user is sent to the routine they
;;      requested.  If an invalid function is requested, the job is
;;      aborted.  The registers are not disturbed.

JSYSX:  EX      (SP),HL                 ; Get PC, save HL
        PUSH    AF                      ; Save current AF
        LD      A,(HL)                  ; Get operation code
        LD      (JSYSOP),A              ; Save it for later
        PUSH    HL                      ; Save PC on stack incase invalid
        CP      JSYSJN                  ; Are we out of range?
        JP      NC,CRASH                ; Yes, assume we crashed
        POP     HL                      ; Get PC back
        POP     AF                      ; Get original AF back
        INC     HL                      ; Bump past operation code
        EX      (SP),HL                 ; Restore PC & HL

;;      The function code has been saved and its range has been checked
;;      All registers at this point are back to normal (temporarily)

        PUSH    HL                      ; Save HL
        PUSH    AF                      ; Save AF again
        LD      A,(JSYSOP)              ; Get the operation code
        ADD     A                       ; *2 for jump table
        LD      HL,JSYSJP               ; Point to jump table
        CALL    ADDH                    ; Point to actual entry
        POP     AF                      ; Get AF back
        CALL    LOADHH                  ; Get routine address
        EX      (SP),HL                 ; Save it on stack, get HL back
        RET                             ; Go to the routine
```

```
        SUBTTL  *** General Purpose Subroutines ***

;;      Add A to the contents of HL
;;      HL=HL+A

ADDH:   ADD     L                       ; Add L
        LD      L,A                     ; Resave it
        RET     NC                      ; Return if no overflow
        INC     H                       ; Adjust H
        RET                             ; & return

;;      Convert the ASCII decimal string
;;      at DE to a binary value in HL
;;      Entry:  DE points to string
;;      Exit:   HL contains value
;;              A contains break character

DECIN:  CALL    NTSSCN                  ; Skip spaces
        LD      HL,0                    ; Start with 0
DECINL: LD      A,(DE)                  ; Get a character
        CALL    NUMER                   ; Is it numeric?
        RET     C                       ; No, return
        INC     DE                      ; Bump character pointer
        SUB     '0'                     ; Convert to binary
        PUSH    BC                      ; Save BC
        LD      B,H                     ; Make copy of
        LD      C,L                     ; HL into BC
        ADD     HL,HL                   ; *2
        ADD     HL,HL                   ; *4
        ADD     HL,BC                   ; *5
        ADD     HL,HL                   ; *10
        POP     BC                      ; Restore BC
        CALL    ADDH                    ; Add in new digit
        JR      DECINL                  ; and go for next

;;      Check if End Of Line, error if not
;;      This routine is also BATCH continuation processor

EOLCHK: PUSH    AF                      ; Save AF
        CALL    NTSSCN                  ; Check for non-space
        LD      (BATPNT),DE             ; Save pointer for BATCH
        OR      A                       ; Is it the end?
        JR      Z,EOLCK1                ; Yes, restore AF & return
        CP      ';'                     ; Is it a comment?
        JR      Z,EOLCK1                ; Yes, restore AF & return
        SUB     MXCHR                   ; Is it multi-command?
        JP      NZ,CMDERR               ; No, error
        INC     DE                      ; Yes, bump past it
        LD      (BATPNT),DE             ; Save new batch pointer
        LD      (BATERR),A              ; Say no BATCH errors
        CPL                             ; Get all 1s
        LD      (BATFLG),A              ; Say BATCH job active
EOLCK1: POP     AF                      ; Get AF back
        RET                             ; & return
        FORM

;;      Convert ASCII HEX value in A to binary
;;      Entry:  A contains character
;;      Exit:   A contains binary value

HBCNV:  SUB     '0'                     ; Convert to binary
        CP      10                      ; Was it "A-F"?
        RET     C                       ; No, return
        SUB     7                       ; Convert for 10-15
        RET                             ; & return

;;      Verify if character in A is valid ASCII HEX
;;      Entry:  A contains character
;;      Exit:   A contains character adjusted for UPPERCASE
;;              Carry flag reset if OK, else
;;              Carry flag set if not OK

HEX:    CALL    NUMER                   ; Check if numeric
        RET     NC                      ; Yes, return
        CALL    LCUC                    ; Convert to uppercase if alpha
        CP      'A'                     ; Is it <"A"?
        RET     C                       ; Yes, return
        CP      'F'+1                   ; Is it >"F"?
        CCF                             ; (Adjust flag)
        RET                             ; Who knows?
        FORM
;;      Print a space, then print binary
;;       value in A to console as HEX
;;      Entry:  A contains value
;;      Register A gets destroyed

HEXBO:  PUSH    AF                      ; Save HEX value
        CALL    SPACE                   ; Print the space
        POP     AF                      ; Get HEX value back
        JR      HEXOUT                  ; & print it

;;      Convert the ASCII HEX string
;;      at DE to a binary value in HL
;;      Entry:  DE points to string
;;      Exit:   HL contains value
;;              A contains break character
```

```
HEXIN:  CALL    NTSSCN                  ; Skip spaces
        LD      HL,0                    ; Start with 0
HEXINL: LD      A,(DE)                  ; Get a character
        CALL    HEX                     ; Valid HEX?
        RET     C                       ; No, return
        INC     DE                      ; Bump ASCII pointer
        CALL    HBCNV                   ; Convert character to binary
        ADD     HL,HL                   ; Make room for new digit
        ADD     HL,HL                   ; by shifting 4 bits
        ADD     HL,HL                   ; which is like
        ADD     HL,HL                   ; multiplying by 10H
        ADD     L                       ; Combine new digit
        LD      L,A                     ; & resave it
        JR      HEXINL                  ; Go for next digit
        FORM

;;      Print 2 byte binary value in HL as HEX
;;      Entry:  HL contains value
;;      Register A gets destroyed

HEXWO:  LD      A,H                     ; Get high byte
        CALL    HEXOUT                  ; Print it
        LD      A,L                     ; Get low byte
                                        ; & drop to printing it

;;      Print binary value in A to console as HEX
;;      Entry:  A contains value
;;      Register A gets destroyed

HEXOUT: PUSH    AF                      ; Save byte for later
        RRCA                            ; Get
        RRCA                            ; left
        RRCA                            ; side
        RRCA                            ; nybble
        CALL    HEXOU1                  ; Print it
        POP     AF                      ; Get byte back
HEXOU1: AND     1111B                   ; Mask off for right nybble
        ADD     '0'                     ; Convert to ASCII
        CP      '9'+1                   ; Is it "A-F"?
        JP      C,OUTCHR                ; No, go print it
        ADD     7                       ; Convert to "A-F"
        JP      OUTCHR                  ; & print it

;;      If character in A is lowercase ASCII,
;;      then convert to UPPERCASE ASCII

LCUC:   CP      'A'+40Q                 ; Is it <"a"?
        RET     C                       ; Yes, return
        CP      'Z'+41Q                 ; Is it >"z"?
        RET     NC                      ; Yes, no conversion
        SUB     40Q                     ; Convert to uppercase
        RET                             ; & return

;;      Load HL with that pointed to by HL
;;      HL=(HL)

LOADHH: PUSH    AF                      ; Save AF
        LD      A,(HL)                  ; Get low byte
        INC     HL                      ; Bump pointer
        LD      H,(HL)                  ; Get high byte
        LD      L,A                     ; Shuffle low byte
        POP     AF                      ; Restore AF
        RET                             ; & return
        FORM

;;      Skip past TABs & spaces & only 1 comma
;;      Entry:  DE points to string
;;      Exit:   DE points past TABs, spaces & comma

NCMSCN: CALL    NTSSCN                  ; Skip TABs and spaces
        CP      ','                     ; Do we have a comma?
        RET     NZ                      ; No, return
        INC     DE                      ; Point past comma
                                        ; Drop to look for next non-space

;;      Skip past TABs & spaces
;;      Do while (DE)=[" "|TAB]: DE=DE+1; Loop
;;      Entry:  DE points to string
;;      Exit:   DE points past TABs & spaces

NTSSCN: LD      A,(DE)                  ; Get a character
        CP      ' '                     ; Is it a space?
        JR      Z,NTSSC1                ; Yes, skip it
        CP      TAB                     ; Is it a <TAB>?
        RET     NZ                      ; No, return
NTSSC1: INC     DE                      ; Bump pointer
        JR      NTSSCN                  ; & keep looking

;;      Verify if character in A is valid ASCII numeric
;;      Entry:  A contains character
;;      Exit:   A contains character
;;              Carry flag reset if OK, else
;;              Carry flag set if not OK
```

```
NUMER:  CP      '0'                     ; Is it <"0"?
        RET     C                       ; Yes, return
        CP      '9'+1                   ; Is it >"9"?
        CCF                             ; (Adjust flag)
        RET                             ; Who knows?

;;      Move print head over to column specified in reg B
;;      Entry:  B contains column number

PRITAB: PUSH    AF                      ; Save AF
PRITB1: LD      A,(TTYPHP)              ; Get current position
        CP      B                       ; Are we where we want to go?
        JP      NC,PPRET                ; Yes, return
        CALL    SPACE                   ; No, move over a column
        JR      PRITB1                  ; Keep moving until done

;;      Clear a block of memory at HL for length in A
;;      Do while A: (HL)=0; HL=HL+1; A=A-1; Loop

ZERO:   LD      (HL),0                  ; Load a 0 into memory
        INC     HL                      ; Bump pointer
        DEC     A                       ; Check count
        JR      NZ,ZERO                 ; & go until done
        RET                             ; then return
        SUBTTL  *** Numeric Argument Processing Routines ***

;;      Entry:  DE points to argument string,
;;                leading spaces & tabs are ignored
;;              Certain defaults MAY be allowed in BC
;;                and HL, depending on routine called
;;      Exit:   DE points just past argument string,
;;                except in the case of ARG3Q, then
;;                DE contains argument 3
;;              BC contains Swath, if required
;;                (argument 2 minus argument 1)
;;              HL contains argument 1
;;              A has the character breaking the string
;;              for ARGx and ARGxD carry flag SET
;;                indicates no argument given

;;      ARGxx   Get argument(s)
;;      xxx1x   HL=arg
;;      xxx2x   HL=arg1, BC=arg2-arg1
;;      xxx2x   HL=arg1, BC=Swath
;;      xxx3Q   HL=arg1, BC=arg2-arg1, DE=arg3
;;      xxx3Q   HL=arg1, BC=Swath, DE=arg3
;;      xxxxQ   If no arg then error
;;      xxxxD   If no arg then default
;;      xxxxQ or xxxxD
;;              If no EOL then error

;;      Get 1 argument, defaults not allowed

ARG1Q:  CALL    ARG1D                   ; Get number
ARGCMC: RET     NC                      ; Number given, return
ARGCME: JP      CMDERR                  ; None given, error

;;      Get 1 argument, defaults allowed

ARG1D:  CALL    ARGH                    ; Get (maybe) argument
ARGEOL: JP      EOLCHK                  ; & go check for EOL

;;      Get 2 arguments, defaults not allowed

ARG2Q:  CALL    ARG2D                   ; Get 2 arguments
        JR      ARGCMC                  ; Check if any given

;;      Get 2 arguments, defaults allowed

ARG2D:  CALL    ARG2                    ; Get 2 arguments
        JR      ARGEOL                  ; & go check EOL
        FORM

;;      Get 3 arguments, 1st argument default allowed,
;;      Swath & 3rd argument defaults not allowed

ARG3Q:  CALL    ARG2                    ; Get first 2 arguments
        JR      C,ARGCME                ; Error if none given
        PUSH    HL                      ; Save 1st argument
        CALL    ARGH                    ; Get 3rd argument
        CALL    EOLCHK                  ; Check for EOL
        EX      DE,HL                   ; Move 3rd argument into place
        POP     HL                      ; Restore 1st argument
        JR      ARGCMC                  ; Go check if all args given

;;      Get 2 arguments, defaults allowed, EOL not checked for

ARG2:   CALL    ARGH                    ; Get 1st argument
                                        ; Drop through to get Swath

;;      Get Swath operator, default allowed
;;      Entry:  DE points to string
;;              HL contains argument 1
;;      Exit:   BC contains Swath
;;                (argument 2 minus argument 1)
;;              DE points just past string
;;              HL contains argument 1
```

```
ARGS:   CALL    NCMSCN                  ; Skip to argument
        CALL    LCUC                    ; Check for lowercase
        CP      'S'                     ; Is it a true Swath?
        PUSH    HL                      ; (Save argument 1)
        JR      NZ,ARGS1                ; No, generate it artificially
        INC     DE                      ; Bump past "S"
        CALL    ARGH                    ; Get the Swath
        JR      C,ARGCME                ; Hey! No fair, "S" with no number
        LD      B,H                     ; Move Swath
        LD      C,L                     ; from HL to BC
ARGSR:  POP     HL                      ; Restore argument 1
        RET                             ; & return
ARGS1:  CALL    ARGH                    ; Get argument 2
        JR      C,ARGSR                 ; None, use default
        POP     BC                      ; Get a copy of
        PUSH    BC                      ; argument 1 into BC
        OR      A                       ; Generate Swath by subtracting
        SBC     HL,BC                   ; argument 1 from argument 2
        INC     HL                      ; Adjust Swath for the end points
        LD      B,H                     ; Move Swath
        LD      C,L                     ; into BC
        POP     HL                      ; Restore argument 1
        LD      A,(DE)                  ; Get breaking character
        OR      A                       ; Say we got the Swath
        RET                             ; & return
        FORM

;;      Numeric Argument Radix Conversion Routine
;;      used by Argument Processing Routines
;;      Entry:  DE points to string
;;              HL contains default
;;      Exit:   DE points just past string
;;              HL contains value (or default if none given)
;;              A contains break character

ARGH:   CALL    NCMSCN                  ; Look for non-space
        PUSH    DE                      ; Save the pointer
        PUSH    HL                      ; Save any default
        CALL    HEXIN                   ; Check for breaker
        CP      '.'                     ; Is it decimal point?
        POP     HL                      ; Restore default
        POP     DE                      ; & pointer
        LD      A,(DE)                  ; Get first character
        JR      Z,ARGH1                 ; Yes, it was decimal point
        CALL    HEX                     ; Check if valid HEX
        RET     C                       ; No, return with default
        CALL    HEXIN                   ; Get new value
        CP      'H'                     ; Is it terminated by "H"?
        JR      Z,ARGH2                 ; Yes, skip over the "H"
        OR      A                       ; Reset cy flag
        RET                             ; & return
ARGH1:  CALL    NUMER                   ; Is first character numeric?
        RET     C                       ; No, return with default
        CALL    DECIN                   ; Get new decimal value
ARGH2:  INC     DE                      ; Bump past "."
        LD      A,(DE)                  ; Get new breaker
        OR      A                       ; Reset cy flag
        RET                             ; & return

        SUBTTL  *** Advanced Console I/O Routines ***

;;      INput a buffered LINE of characters
;;      Entry:  DE points to line buffer
;;              A contains maximum length
;;      Exit:   DE points to line buffer
;;              A contains actual entered length
;;              Line is terminated with binary 0

INLINE: PUSH    BC                      ; Save BC
        PUSH    HL                      ; & HL also
        LD      C,A                     ; Save maximum length
INLING: LD      B,0                     ; Initial length of 0
        LD      H,D                     ; Point HL to
        LD      L,E                     ; line buffer
INLINL: CALL    TTYGET                  ; Get a character
        JR      Z,INLINL                ; If <NUL>, then ignore it
        CP      CTLE                    ; Is it Control-E?
        CALL    Z,CRLF                  ; If so, go to new line
        JR      Z,INLINL                ; then ignore it
        CP      CR                      ; Is it <CR>?
        JR      Z,INLINB                ; Yes, end of line
        CP      BS                      ; Is it backspace?
        JR      Z,INLIN1                ; Yes, delete a character
        CP      DEL                     ; Is it <DEL>?
        JR      NZ,INLIN2               ; No, check something else
INLIN1: XOR     A                       ; Zap A
        OR      B                       ; To check buffer length
        JR      Z,INLINL                ; Empty, nothing to delete
        DEC     HL                      ; Drop pointer
        DEC     B                       ; & count
        CALL    BACKSP                  ; Blank out character
        LD      A,(HL)                  ; Get what character that was
        CP      ' '                     ; Was it a control character?
        CALL    C,BACKSP                ; Yes, blank out the "^"
        JR      INLINL                  ; Go get another character
```

```
INLIN2: PUSH    AF                      ; Save character
        CALL    OUTECH                  ; Print it
        POP     AF                      ; Restore it
        CP      CTLC                    ; Is it ^C?
        JR      Z,INLIN3                ; Yes, abort the job
        CP      CTLZ                    ; Is it ^Z?
INLIN3: CALL    Z,CRLF                  ; Go to next line if so
        JP      Z,REENTC                ; Yes, go back to command level
        CP      CTLU                    ; Is it ^U?
        CALL    Z,CRLF                  ; Go to next line if so
        JR      Z,INLING                ; Yes, scrap this line & get another
        LD      (HL),A                  ; Save the character
        INC     HL                      ; Bump pointer
        INC     B                       ; & count
        LD      A,C                     ; Get maximum length
        CP      B                       ; Are we there yet?
        JR      NZ,INLINL               ; No, go for another character
INLINB: LD      (HL),0                  ; Zap final byte
        LD      A,B                     ; Get final length
        POP     HL                      ; Get HL back
        POP     BC                      ; and BC also
        OR      A                       ; Set length flags
                                        ; Go to next line & return

;;      Print a <CR><LF> sequence

CRLF:   CALL    OUTSTI                  ; Print the following
        DM      CR                      ; <CR> will also make <LF>
        RET                             ; Return

;;      Print a "<BS><SPACE><BS>" sequence

BACKSP: CALL    OUTSTI                  ; Use a short cut
        DM      '\B \B'                 ; "<BS> <BS>"
        RET                             ; Done

;;      Echo character from input routine
;;      Entry:  A contains the character
;;      Exit:   A gets destroyed

OUTECH: CP      ' '                     ; Check if graphic character
        JR      NC,OUTCHR               ; Yes, go print it
        CP      CR                      ; Check if <CR>
        JR      Z,OUTCHR                ; Yes, go print it
        PUSH    AF                      ; Save character
        LD      A,'^'                   ; Get "^"
        CALL    OUTCHR                  ; Print it
        POP     AF                      ; Get character back
        ADD     'A'-1                   ; Convert to graphic
        JR      OUTCHR                  ; Print it & return

;;      Get a processed (echoed) character
;;      Exit:   A contains the character

INCHR:  CALL    TTYGET                  ; Get a character
        CP      DEL                     ; Is it <DEL>?
        RET     Z                       ; Yes, don't echo it
        CP      ' '                     ; Is it printable?
        JR      NC,OUTCHR               ; Yes, print it
        CP      CR                      ; Is it <CR>?
        JR      Z,OUTCHR                ; Yes, print it
        RET                             ; No, return
        FORM

;;      Print a space

SPACE:  LD      A,' '                   ; Load the space
                                        ; & drop to OUTCHR

;;      Output a processed character via TTYOUT
;;      & check for characters typed. Also adjust
;;      console position & check for BATCH errors
;;      Entry:  A contains character
;;      All registers preserved unless output is aborted

OUTCHR: PUSH    AF                      ; Save character & flags
        AND     ^^7                     ; Mask off parity
        PUSH    HL                      ; Save HL
        LD      HL,TTYPHP               ; Point to cursor position
        CP      BEL                     ; Is the character <BEL>?
        JR      NZ,OUTCH0               ; No, check what else
        LD      A,(HL)                  ; Get the console position
        OR      A                       ; Is it 0?
        JR      NZ,OUTCHQ               ; No, this is just another ding
        CPL                             ; Yes, this means an error
        LD      (BATERR),A              ; Set error flag
OUTCHQ: LD      A,BEL                   ; Reload the <BEL>
OUTCH0: CP      BS                      ; Is it <BS>?
        JR      NZ,OUTCH1               ; No, check for <TAB>
        DEC     (HL)                    ; Back up cursor pointer
        JR      OUTCHG                  ; Go print the <BS>
OUTCH1: CP      TAB                     ; Is it <TAB>?
        JR      NZ,OUTCH3               ; No, check for graphic
```

```
OUTCH2: CALL    SPACE                   ; Print a space
        LD      A,111B                  ; Load mask for <TAB> position
        AND     (HL)                    ; Are we there yet?
        JR      NZ,OUTCH2               ; No, keep printing
        JR      OUTCHC                  ; Yes, return
OUTCH3: CP      DEL                     ; Is it <DEL>?
        JR      Z,OUTCHG                ; Yes, just print it
        CP      ' '                     ; Is it graphic?
        JR      C,OUTCHG                ; No, just print it
        INC     (HL)                    ; Yes, Bump cursor pointer
OUTCHG: CALL    TTYOUT                  ; Print the character
        CP      CR                      ; Is it <CR>?
        JR      NZ,OUTCHC               ; No, check if can be aborted
        LD      (HL),0                  ; Yes, say cursor position at 0
        LD      A,LF                    ; Load a <LF>
        CALL    OUTCHR                  ; Echo it also
OUTCHC: POP     HL                      ; Restore HL
        CP      LF
        JR      NZ,PPRET                ; No, restore character & return
        CALL    TTSCAN                  ; Is anything waiting for us?
        JR      Z,PPRET                 ; No, return
        CALL    TTYGT1                  ; Get what it was
        CP      CTLS                    ; Is it pause?
        CALL    Z,TTYGT1                ; Yes, wait for something
        CP      CTLC                    ; Control-C?
        JP      Z,INLIN2                ; Yes, echo it & go back to command
        CP      CTLZ                    ; Control-Z?
        JP      Z,INLIN2                ; Yes, echo it & go back to command
        CP      ESC                     ; Is it <ESC> to abort?
        JP      Z,REENTQ                ; Yes, go back to command level
        CP      ALT                     ; Check for <ALT>
        JP      Z,REENTQ                ; Yes, go back to command level
        CP      ' '                     ; Is it printable?
        JR      C,PPRET                 ; No, don't save it
        LD      (CHRBUF),A              ; Yes, save it for later
PPRET:  POP     AF                      ; Restore character
        RET                             ; & return

;;      Print string pointed to by HL
;;      String terminates on either 0 or ^7
;;      Entry:  HL points to string

OUTSTR: PUSH    AF                      ; Save AF
OUTST1: LD      A,(HL)                  ; Get a character
        INC     HL                      ; Bump pointer
        OR      A                       ; Is it he end?
        JR      Z,OUTST2                ; Yes, return
        CALL    OUTCHR                  ; Print the character
        JP      P,OUTST1                ; If ^7 then go print more
OUTST2: POP     AF                      ; Restore AF
        RET

;;      Print string immediately following CALL
;;      String terminates on either 0 or ^7
;;      Entry:  string is immediately after CALL

OUTSTI: EX      (SP),HL                 ; Get string pointer (PC)
        CALL    OUTSTR                  ; Print the string
        EX      (SP),HL                 ; Save PC back on stack
        RET                             ; & return
        FORM

;;      Get a single (advance) character
;;      Exit:   A contains the character

TTYGET: PUSH    HL                      ; Save HL
        LD      HL,CHRBUF               ; Point to character buffer
        LD      A,(HL)                  ; Get whatever
        LD      (HL),0                  ; Zap the buffer
        POP     HL                      ; Restore HL
        OR      A                       ; Was there anything?
        RET     NZ                      ; Yes, return
TTYGT1: CALL    TTYIN                   ; Get a byte
        AND     ^7                      ; Mask off parity
        RET                             ; & return

;;      Check if advance character ready
;;      Exit:   A contains 0 if none, Z flag is set
;;              A contains -1 if any, Z flag is reset

SCNCHR: LD      A,(CHRBUF)              ; Get whatever in character buffer
        OR      A                       ; Was there anything?
        LD      A,-1                    ; Get all ones incase so
        RET     NZ                      ; Yes, return
                                        ; No, check if hardware ready

        SUBTTL  *** Primitive Console I/O Routines ***

;;      Check if byte ready from current console
;;      Exit:   A contains 0 if none, Z flag is set
;;              A contains -1 if any, Z flag is reset
```

```
TTSCAN: IN      A,HPORT                 ; Get status
        AND     HRDA                    ; Check if anything there
        RET     Z                       ; Nothing, return
        LD      A,-1                    ; Load all ones
        RET                             ; & return

;;      Get a byte from current console
;;      Exit:   A contains the byte

TTYIN:  CALL    TTSCAN                  ; Check if anything there
        JR      Z,TTYIN                 ; No, keep waiting
        IN      A,HPORT+1
        RET                             ; & return

;;      Output a byte to current console
;;      Entry:  A contains the byte
;;      All registers preserved except F

TTYIO:  CALL    TTYIN
TTYOUT: PUSH    AF                      ; Save the character
TTYOU1: IN      A,HPORT                 ; Get the status
        AND     HTBE                    ; Is it ready yet?
        JR      Z,TTYOU1                ; No, wait for ready
        POP     AF                      ; Get character back
        OUT     HPORT+1,A               ; Send the character
        RET                             ; then return

CONGEN: EQU     $-1
        FORM


;;      Command Routine Jump Table

CMDJMP: DW      BATCH
        DW      CMDERR
        DW      CMDERR
        DW      CMDERR
        DW      DISPLY
        DW      EXAMIN
        DW      CMDERR
        DW      GOTO
        DW      CMDERR
        DW      CMDERR
        DW      CMDERR
        DW      CMDERR
        DW      CMDERR
        DW      MOVE
        DW      NULLS
        DW      OUTPUT                  ; O - Output to Port
        DW      CMDERR                  ; P - Error
        DW      FIND                    ; Q - Query Memory
        DW      READ                    ; R - Read Binary Data
        DW      SETMEM                  ; S - Set Memory
        DW      CMDERR                  ; T - Error
        DW      CMDERR                  ; U - Error
        DW      VERIFY                  ; V - Verify Memory
        DW      WRITE                   ; W - Write Binary Data
        DW      CMDERR                  ; X - Error
        DW      CMDERR                  ; Y - Error
        DW      ZAP                     ; Z - Zap Memory

        SUBTTL  *** JSYS Entry Jump Table ***

;;      JSYS Entry Jump Table
;;      This table is used for user entry processing;
;;      Each of the addresses listed below
;;      corresponds to a user function.

JSYSJP: DW      REENTE                  ; Re-enter the monitor
        DW      TTYIN                   ; Input a pure byte
        DW      TTSCAN                  ; Check for pure byte ready
        DW      TTYOUT                  ; Output a pure byte
        DW      INLINE                  ; Input a buffered line
        DW      INCHR                   ; Input a processed character
        DW      SCNCHR                  ; Check for character ready
        DW      OUTCHR                  ; Output a processed character
        DW      OUTSTR                  ; Output a string
        DW      OUTSTI                  ; Output immediate string
        DW      CRLF                    ; Output a <CR><LF> sequence
        DW      CONGEN                  ; Get console hardware info
        DW      BATCH1                  ; Set up a batch job
        DW      ARGH                    ; Convert a single argument
        DW      ARG2                    ; Convert two arguments
        DW      HEXOUT                  ; Display number in HEX
JSYSJN  EQU     [$-JSYSJP]/2            ; Total # of functions

        SUBTTL  *** Constants & Lookup Tables ***


        ORG     0FF00H          ; Variables at the top page of RAM

VARBS   EQU     $                       ; Reference for beginning of variables

BREAK:  DS      3                       ; DEBUG break transfer (JP nn)

JSYSOP: DS      1                       ; Current JSYS operation code

DISPNT: DS      2                       ; Last DM command address
SETPNT: DS      2                       ; Last SM command address
```

```
CONNUM: DS      1               ; Current console #
                                ; If minus then Host else
                                ; if 040H then CSP else
                                ; is Quadart channel number
TTYPHP: DS      1               ; Console print position
CHRBUF: DS      1               ; Advance console character buffer
CONTMP: DS      1               ; Temporary storage for console #

XCTADR: DS      2               ; Return execution address

BATDAT  EQU     $               ; Reference for batch data block
BATFLG: DS      1               ; Indicates batch job in progress
BATERR: DS      1               ; Indicates error occurred
BATPNT: DS      2               ; Batch command line pointer

        DS      1               ; File operations abort inhibit flag

LINLEN  EQU     72              ; Console input line length
LINBUF: DS      LINLEN+1        ; Console input line buffer

STACK   EQU     0FFE0H          ; Where to load Stack Pointer

XXXXXX  EQU     STACK           ; User available free space

        SUBTTL  *** Equates ***

;;      I/O Port Equates

HPORT   EQU     00H
HRDA    EQU     ^6              ; Host Receive Data Available
HTBE    EQU     ^7              ; Host Transmitter Buffer Empty
                                ;       LsB/MsB   ll
                                ;        Square     011
                                ;        Binary       0

        FORM

;;      ASCII Equates

CTLC    EQU     003             ; Control-C for job abort
CTLE    EQU     005             ; Control-E for physical <CR><LF>
BEL     EQU     007             ; Bell
BS      EQU     008             ; Backspace
TAB     EQU     009             ; Horizontal Tab
LF      EQU     010             ; Line Feed
CR      EQU     013             ; Carriage Return
CTLS    EQU     019             ; Control-S for pause
CTLU    EQU     021             ; Control-U for delete line
CTLZ    EQU     026             ; Control-Z for job abort
ESC     EQU     027             ; Escape
MXCHR   EQU     '\\'            ; Multiple command terminator
ALT     EQU     125             ; Altmode
DEL     EQU     127             ; Delete character

        END     START
```

```
        TITLE   GPIB Down Loader
        SUBTTL  *** Setting Things Up & Doing The Transfer ***
        NAME    GPIBLD
;       12:00   10-Nov-80

VERNUM  EQU     01                      ; Version #
RLSNUM  EQU     08                      ; Release #

        ORG     100H                    ; Start at standard TPA
        JP      START                   ; Skip the following

IPORT:  DB      080H                    ; GPIB Control Port
        DW      0EDEDH                  ; CROMIX version ID code
        DB      RLSNUM,VERNUM           ; Actual version number

;       Main signon message
SIGNON: DB      'GPIB  Down Loader - version '
        DB      VERNUM/10+'0',VERNUM%10+'0','.'
        DB      RLSNUM/10+'0',RLSNUM%10+'0','\R\N$'

;       Set up stack & print signon
START:  LD      HL,(ENTRY+1)            ; Point to highest user RAM
        LD      SP,HL                   ; User it for stack
        LD      DE,SIGNON               ; Point to signon message
        LD      C,CFNSTR                ; CDOS: Print a String
        CALL    ENTRY                   ; Print the signon message

;       Set up variables in case restarted
        LD      BC,KCMDSL               ; Copy constant command
        LD      DE,CMDS                 ; string into variables
        LD      HL,KCMDS                ; incase of restart
        LDIR                            ; Copy made

;       Get & set disk specs
        LD      C,CFNGDX                ; CDOS: Get current disk
        CALL    ENTRY                   ; Fetch
        LD      (CURDSK),A              ; Save it for the return trip
        LD      HL,TFCB                 ; Point to specified disk
        LD      E,(HL)                  ; Get it for later
        LD      (HL),0                  ; Right now zap it out
        DEC     E                       ; Was there one given?
        LD      C,CFNSDX                ; CDOS: Set new disk
        CALL    P,ENTRY                 ; Yes, go set it up

;       Misc. initialization
        CALL    INIT                    ; Grab the GPIB's attention
        CALL    ADRSET                  ; Set up the load address
        FORM

;       Look for & set up the file
FILE1:  LD      A,(TFCB+1)              ; Get 1st character of filename
        CP      '?'                     ; Is it wild?
        LD      DE,ILWERM               ; Load error message just in case
        JP      Z,EXITP                 ; Yes, print message & abort
        LD      C,CFNLKU                ; CDOS: Lookup a file
        LD      DE,TFCB                 ; Point to FCB
        CALL    ENTRY                   ; Go get it
        INC     A                       ; Is it there?
        JR      NZ,FILE2                ; Yes, continue
        LD      A,(TFCB+9)              ; Get 1st character of extension
        CP      ' '+1                   ; Is there one?
        LD      DE,FNFERM               ; Error message just in case
        JR      NC,EXITP                ; Yes, no retries allowed
        LD      BC,3                    ; Default extension 3 characters
        LD      DE,TFCB+9               ; Point to where extension goes
        LD      HL,KEXT                 ; Point to constant extension
        LDIR                            ; Copy it in
        JR      FILE1                   ; Go try to get the file again
FILE2:  LD      A,(TFCB+15)             ; Get the # of records
        OR      A                       ; Are there any?
        LD      DE,EMPERM               ; Error message incase empty
        JR      Z,EXITP                 ; No, print message & abort
        FORM

;       Transfer the disk file to the GPIB
LOOP:   LD      C,CFNRD                 ; CDOS: Read a record
        LD      DE,TFCB                 ; Point to FCB
        CALL    ENTRY                   ; Get the record
        OR      A                       ; End of file?
        JR      NZ,EXIT                 ; Yes, stop
        LD      HL,(LDADDR)             ; Get the load address
        PUSH    HL                      ; Save it for later
        LD      DE,0FF00H               ; Check if running
        SBC     HL,DE                   ; into system memory
        POP     HL                      ; Get load address back
        LD      DE,FTLERM               ; Load error message just in case
        JR      NC,EXITP                ; Yes, print message & stop
        LD      DE,CMDS1                ; Point to command string
        LD      A,H                     ; Get high load address
        CALL    HEXASC                  ; Convert the byte to ASCII
        LD      A,L                     ; Get low load address
        CALL    HEXASC                  ; Convert to ASCII
        LD      DE,80H                  ; Point to next block
        ADD     HL,DE                   ; to be loaded
        LD      (LDADDR),HL             ; Save it for next time
        LD      B,KCMDSL                ; Get command string length
        LD      HL,CMDS                 ; Point to command string
```

```
LOOP1:  LD      A,(HL)                  ; Get a character
        INC     HL                      ; Bump pointer
        CALL    SEND                    ; Send the character
        CALL    BUCKET                  ; Catch any echo
        DJNZ    LOOP1                   ; Go until all string sent
        LD      HL,TBUF                 ; Point to data buffer
LOOP2:  LD      A,(HL)                  ; Get a byte
        CALL    SEND                    ; Send it to the GPIB
        INC     L                       ; Bump pointer
        JR      NZ,LOOP2                ; Go until end of buffer
        CALL    BUCKET                  ; Catch prompt echo
        JR      LOOP                    ; Go get next record

        SUBTTL  *** Special Subroutines ***

;       Parse & set up address from TFCB2
ADRSET: LD      DE,TFCB2+1              ; Point to TFCB #2 ASCII
        LD      HL,DEFLDA               ; Default load address
        CALL    HEXIN                   ; Go get the address
        LD      (LDADDR),HL             ; Save it for later
        CP      ' '+1                   ; Broke with a space?
        RET     C                       ; Yes, return
ADRST1: LD      DE,ILAERM               ; Bad address argument
                                        ; Drop through to print message

;       Exit routines
EXITP:  LD      C,CFNSTR                ; CDOS: Print a String
        CALL    ENTRY                   ; Print the whatever
EXIT:   LD      A,(CURDSK)              ; Get command disk
        LD      C,CFNSDX                ; CDOS: Set Disk
        LD      E,A                     ; Save disk in E
        CALL    ENTRY                   ; Set it up in CDOS
        JP      ABORT                   ; Stop the job

;       Initialize communications with the GPIB
INIT:   LD      A,(IPORT)               ; Get the GPIB address
        LD      C,A                     ; Save it in C
        IN      A,(C)                   ; Make sure at least 1 bit is 0
        IN      A,(C)                   ; Get GPIB status
        CP      -1                      ; Is there an GPIB there?
        LD      DE,NQSERM               ; Load message just in case
        JR      Z,EXITP                 ; No GPIB, error
        LD      B,24                    ; Attention retry count
INIT1:  LD      A,CR                    ; Send a <CR>
        CALL    SEND                    ; to initialize
        XOR     A                       ; Delay a full byte
INIT2:  DEC     A                       ; to allow the GPIB
        JR      NZ,INIT2                ; to respond
        IN      A,(C)                   ; Get the GPIB status
        AND     RDA                     ; Check if anything ready
        JR      Z,INITC                 ; No, Try again
        INC     C                       ; Point to GPIB data port
        IN      A,(C)                   ; Get whatever
        DEC     C                       ; Point to status port again
        AND     ~^7                     ; Mask off parity
        CP      CR                      ; Is it a <CR>?
        JR      Z,BUCKET                ; Yes, we got it, ignore the rest
INITC:  DJNZ    INIT1                   ; Drop retry count & try again
        LD      DE,ITOERM               ; Error, GPIB not responding
        JR      EXITP                   ; Print error & stop job

;       Send a byte to the GPIB
SEND:   PUSH    BC                      ; Save BC (use B for timeout)
        PUSH    AF                      ; Save byte
        LD      B,0                     ; Timeout of 256 retries
        LD      A,(IPORT)               ; Get GPIB status port
        LD      C,A                     ; Save it in C
SEND1:  IN      A,(C)                   ; Get the GPIB status
        AND     TBE                     ; Is it ready for a byte?
        JR      NZ,SEND2                ; Yes, send it & return
        DJNZ    SEND1                   ; No, check timeout count
        LD      DE,ITOERM               ; Oops, GPIB crashed
        JR      EXITP                   ; Print error message & abort
SEND2:  POP     AF                      ; Get byte back
        INC     C                       ; Point to data port
        OUT     (C),A                   ; Send the byte
        POP     BC                      ; Restore BC
        RET                             ; & return

;       GPIB byte bucket
BUCKET: LD      A,(IPORT)               ; Get the GPIB status port
        LD      C,A                     ; Save it in C
        LD      A,200                   ; Delay 200 loops
BUCKE1: DEC     A                       ; to allow the GPIB
        JR      NZ,BUCKE1               ; to bounce back
        IN      A,(C)                   ; Get the GPIB status
        AND     RDA                     ; Is anything there?
        RET     Z                       ; No, return
        INC     C                       ; Point to data port
        IN      A,(C)                   ; Get & ignore what's there
        DEC     C                       ; Point to status port again
        JR      BUCKET                  ; Go for next byte
```

```
        SUBTTL  *** General Purpose Routines ***

;       Convert ASCII HEX character in A to binary
HBCNV:  SUB     '0'                     ; Drop ASCII bias
        CP      9+1                     ; Greater than 9? (A-F)
        RET     C                       ; No, return
        SUB     7                       ; Adjust for A-F
        RET                             ; Return

;       Verify if character in A is valid ASCII HEX
;       Cy reset indicates OK, else error
HEX:    CALL    NUMER                   ; Is numeric?
        RET     NC                      ; Yes, OK, return
        CP      'A'                     ; Is <"A"?
        RET     C                       ; Yes, return
        CP      'F'+1                   ; Is >"F"?
        CCF                             ; Adjust flag
        RET                             ; Return w/ answer

;       Convert byte in A to ASCII HEX string at DE
HEXASC: PUSH    AF                      ; Save byte for right nybble
        RRCA                            ; Move left
        RRCA                            ; nybble
        RRCA                            ; into
        RRCA                            ; place
        CALL    HEXAS1                  ; Print left nybble
        POP     AF                      ; Get byte back
HEXAS1: AND     1111B                   ; Isolate right nybble
        ADD     '0'                     ; Convert to ASCII
        CP      '9'+1                   ; Is it ASCII numeric?
        JR      C,HEXAS2                ; Yes, save it
        ADD     7                       ; Adjust for A-F
HEXAS2: LD      (DE),A                  ; Save the character
        INC     DE                      ; Bump the pointer
        RET                             ; Return

;       Convert ASCII HEX string at
;       DE to binary number in HL
HEXIN:  LD      A,(DE)                  ; Get a character
        CALL    HEX                     ; Valid ASCII HEX?
        RET     C                       ; No, return w/ default
        LD      HL,0                    ; Start with all 0s
HEXIN1: LD      A,(DE)                  ; Get a character
        CALL    HEX                     ; Valid ASCII HEX?
        RET     C                       ; No, finished
        INC     DE                      ; Bump character pointer
        CALL    HBCNV                   ; Convert character to binary
        ADD     HL,HL                   ; Multiply HL
        ADD     HL,HL                   ; times 16 to
        ADD     HL,HL                   ; make room for
        ADD     HL,HL                   ; the new nybble
        ADD     L                       ; Add the nybble in
        LD      L,A                     ; Resave the lower byte
        JR      HEXIN1                  ; Go for next digit

;       Verify if character in A is valid ASCII numeric
;       Cy reset indicates OK, else error
NUMER:  CP      '0'                     ; Is <"0"?
        RET     C                       ; Yes, return
        CP      '9'+1                   ; Is >"9"?
        CCF                             ; Adjust flag
        RET                             ; Return w/ answer

        SUBTTL  *** Constants & Text ***

;       Command string to the GPIB monitor
KCMDS:  DB      'R    S80',CR           ; "RxxxxS80<CR>"
KCMDSL  EQU     $-KCMDS                 ; Length of string

;       Error messages
NQSERM: DB      'No GPIB in System\R\N$'
ITOERM: DB      'GPIB Not Responding - Timeout\R\N$'
ILAERM: DB      'Illegal Load Address\R\N$'
ILWERM: DB      'Wild File Specification Illegal\R\N$'
FTLERM: DB      'File Too Large\R\N$'
EMPERM: DB      'File is Empty\R\N$'
FNFERM: DB      'File Not Found - Check For Extension ".'

KEXT:   DB      'GPB"\R\N$'             ; Secondary file extension

        SUBTTL  *** Variables & Equates ***

;       Misc. Small Variables
CURDSK: DS      1                       ; Command disk #
LDADDR: DS      2                       ; Current load address

;       Command String Storage - "RxxxxS80<CR>"
CMDS:   DS      1                       ; "R"
CMDS1:  DS      8                       ; "xxxxS80<CR>"

;       CDOS Location Equates
ABORT   EQU     0000H                   ; Program abort location
ENTRY   EQU     0005H                   ; CDOS CALL entry point
TFCB    EQU     005CH                   ; Transient FCB #1
TFCB2   EQU     TFCB+10H                ; Transient FCB #2
TBUF    EQU     0080H                   ; Transient disk buffer
```

```
;       CDOS CALL Codes
CFNSTR  EQU     009                     ; print a STRing
CFNSDX  EQU     014                     ; Set DiSK
CFNLKU  EQU     015                     ; LooKUp a file
CFNRD   EQU     020                     ; ReaD a record
CFNGDX  EQU     025                     ; Get DiSK

;       I/O Equates
HQBIT   EQU     ^5                      ; GPIB is connected to Quadart
RDA     EQU     ^6                      ; GPIB Rx Ready
TBE     EQU     ^7                      ; GPIB Tx Ready

;       ASCII Equates
CR      EQU     013                     ; Carriage Return

;       Things with nowhere else to live
DEFLDA  EQU     4000H                   ; Default load address

        END     START                   ; That's all, folks!
```

```
        TITLE   GPIB Processor Communicator
        SUBTTL  *** Setting Things Up ***
        NAME    GPIBC
;       12:00   10-Nov-80

VERNUM  EQU     01                      ; Version #
RLSNUM  EQU     08                      ; Release #

        ORG     100H                    ; Start at standard TPA
        JP      START                   ; Skip the following

IPORT:  DB      080H                    ; GPIB base address
IPORT1: DB      082H
DEFTRM: DB      'D'-40H                 ; Default terminator
        DW      0EDEDH                  ; CROMIX version ID code
        DW      RLSNUM,VERNUM           ; Actual version number

;       Main signon message
SIGNON: DB      'GPIB/HOST Communicator version '
        DB      VERNUM/10+'0',VERNUM%10+'0','.'
        DB      RLSNUM/10+'0',RLSNUM%10+'0'

;       Text to go to next line
CRLFM:  DB      '\R\N$'                 ; "<CR><LF>$"

;       Set up stack & print signon message
START:  LD      SP,(CDOS+1)             ; Load SP as top of user area
        LD      DE,SIGNON               ; Point to the signon message
        CALL    PRINT                   ; Print the signon message
        FORM

;       Determine terminator character

        LD      A,(DEFTRM)              ; Get the default terminator
        LD      D,A                     ; Save it in D for later
        LD      A,(TFCB+1)              ; Get 1st character of default FCB
        CP      ' '                     ; Is it a space?
        JR      Z,DEFX2                 ; If so, use default terminator
        CP      'A'                     ; Is specified terminator < "A"?
        JR      C,BADTRM                ; Error if so, print message & abort
        CP      '^'+1                   ; Is specified terminator > "^"?
        JR      NC,BADTRM               ; Error if so, print message & abort
        CP      'P'                     ; Is the terminator < "P"?
        JR      C,DEFX1                 ; Yes, assume OK
        CP      'T'                     ; Is terminator now < "T"?
        JR      C,BADTRM                ; Yes, that's an illegal range
DEFX1:  SUB     40H                     ; Zap ASCII bias to get Control CHR
        LD      D,A                     ; Save the terminator in D
DEFX2:  LD      A,(IPORT)               ; Get the GPIB address
        LD      C,A                     ; Save it in C for general use
        LD      A,(IPORT1)
        LD      E,A
        LD      A,D                     ; Get a copy of the terminator in A
        CP      '^'-40H                 ; Was the terminator "^"?
        PUSH    DE                      ; Save the terminator around INIT
        CALL    NZ,INIT                 ; Initialize communications if not
        POP     DE                      ; Get the terminator back

        SUBTTL  *** Do The Actual Communications Transfers ***

;       This loop processes characters from the HOST to the GPIB

ECHO1:  PUSH    BC                      ; Save GPIB address
        LD      C,CFNTTS                ; CDOS: Check for character ready
        CALL    CDOS                    ; Fetch
        OR      A                       ; Is there anything?
        LD      C,CFNTTI                ; CDOS: Get a character (no echo)
        CALL    NZ,CDOS                 ; Yes, get host character
        POP     BC                      ; Get back GPIB address
        OR      A                       ; <NUL>?
        JR      Z,ECHO2                 ; Yes, nothing there
        CP      D                       ; Abort in the middle?
        JR      Z,EXITCR                ; Yes, stop the job
        CP      'Y'-40H
        JR      NZ,EC1
        LD      B,E
        LD      E,C
        LD      C,B
        PUSH    DE
        CALL    INIT
        POP     DE
        LD      A,0DH
        JR      EC2
EC1:    CP      'X'-40H
        JR      NZ,EC2
        LD      B,E
        LD      E,C
        LD      C,B
        LD      A,0DH
EC2:    LD      B,A                     ; Save the character
        IN      A,(C)                   ; Get the GPIB status
        AND     TBE                     ; Ready to receive?
        JR      Z,ECHO2                 ; No, this character is lost
        INC     C                       ; Point to the GPIB data port
        OUT     (C),B                   ; Send the data to the GPIB
        DEC     C                       ; Point to status port again
```

```
;       This loop processes characters from the GPIB to the HOST

ECHO2:  IN      A,(C)                   ; Get GPIB status
        AND     RDA                     ; Anything there?
        JR      Z,ECHO1                 ; No, check host
        INC     C                       ; No, point to the GPIB data port
        PUSH    DE
        IN      E,(C)                   ; Get the GPIB data
        DEC     C                       ; Point to status port again
        PUSH    BC                      ; Save GPIB address
        LD      C,CFNOCH                ; CDOS: Output a character
        CALL    CDOS                    ; Send it to the host
        POP     BC                      ; Get back GPIB address
        POP     DE
        JR      ECHO1                   ; Now go check host

;       This part ignores file commands from the GPIB
;       (so GPIB will "Timeout" instead of hang)

ECHO3:  LD      HL,15000                ; Delay long
ECHO3A: DEC     HL                      ; enough so
        LD      A,H                     ; the GPIB
        OR      L                       ; gets the
        JR      NZ,ECHO3A               ; message
        JR      ECHO1                   ; then proceed as normal

        SUBTTL  *** Termination & Print Routines ***

;       Bad terminator error
;       Print error message then stop

BADTRM: LD      DE,ILTERM               ; Point to error message
                                        ; Drop through to print it

;       Exit routines

EXITP:  CALL    PRINT                   ; Print whatever string

EXITCR: LD      DE,CRLFM                ; Point to new-line text
        CALL    PRINT                   ; Print it
        JP      ABORT                   ; Abort the job

;       Print string pointed to by DE

PRINT:  LD      C,CFNOST                ; CDOS: Print a string
        JP      CDOS                    ; Print the string & return

        SUBTTL  *** Getting Aquainted With The GPIB ***

;       Initialize communications with the GPIB
;       This has the same effect as typing
;       <CR> and waiting for a response

INIT:   IN      A,(C)                   ; Get GPIB status
        IN      A,(C)                   ; Make sure at least 1 bit is 0
        CP      -1                      ; Is an GPIB there?
        LD      DE,NISERM               ; Load message in case not
        JR      Z,EXITP                 ; No, print message & stop
        LD      A,080H
        OUT     (C),A
INIT0:  DEC     A
        JR      NZ,INIT0
        OUT     (C),A
        LD      B,24                    ; Initialize attention retry count
INIT1:  PUSH    BC                      ; Save retry count
        LD      B,0                     ; Timeout of 256 status retries
INIT2:  IN      A,(C)                   ; Read the status port
        AND     TBE                     ; Is it ready for a character ?
        JR      NZ,INIT3                ; Yes, send it and return
        DJNZ    INIT2                   ; No, check timeout count
        JR      INITE                   ; Print error message and abort
INIT3:  POP     BC                      ; Restore retry count
        LD      A,CR                    ; Load a <CR>
        INC     C                       ; Point to data port
        OUT     (C),A                   ; Send the <CR>
        DEC     C                       ; Point back to status port
        XOR     A                       ; Delay a full byte
INIT4:  INC     A                       ; To allow the GPIB
        JR      NZ,INIT4                ; To respond
        IN      A,(C)                   ; Get the GPIB status
        AND     RDA                     ; Check if a character is ready
        JR      Z,INITC                 ; No, try again
        INC     C                       ; Point to data port
        IN      A,(C)                   ; Get character
        DEC     C                       ; Point back to status port
        AND     ~^7                     ; Mask off parity bit
        CP      CR                      ; Is it a <CR>?
        RET     Z                       ; Yes, we got it, return
INITC:  DJNZ    INIT1                   ; Drop retry count and try again
INITE:  LD      DE,ITOERM               ; Error, GPIB not responding
        JR      EXITP

        SUBTTL  *** Error Messages ***

ILTERM: DB      'Illegal Terminator - Must be "A" through "^"\R\N'
        DB      '("P" through "S" not allowed)$'
NISERM: DB      'No GPIB In System$'
ITOERM: DB      'GPIB Not Responding - Timeout$'
```

```
        SUBTTL  *** Misc EQUates ***

ABORT   EQU     0000H           ; Abort back to CDOS
CDOS    EQU     0005H           ; CDOS CALL entry point
TFCB    EQU     005CH           ; Transient FCB

CFNOCH  EQU     002             ; Output a CHaracter
CFNOST  EQU     009             ; Output a STring
CFNTTS  EQU     011             ; TTy Scan for character
CFNTTI  EQU     128             ; TTy Input a character

RDA     EQU     ^6              ; Receive Data Available
TBE     EQU     ^7              ; Transmitter Buffer Empty

CR      EQU     013             ; Carriage Return

        END     START
```

Appendix E

LIMITED WARRANTY

Cromemco, Inc. ("Cromemco") warrants this product against defects in material and workmanship to the original purchaser for ninety (90) days from the date of purchase, subject to the following terms and conditions.

What Is Covered By This Warranty:

During the ninety (90) day warranty period Cromemco will, at its option, repair or replace this Cromemco product or repair or replace with new or used parts any parts or components, manufactured by Cromemco, which prove to be defective, provided the product is returned to an Authorized Cromemco Dealer as set forth below.

How To Obtain Warranty Service:

You should immediately notify IN WRITING your Authorized Cromemco Dealer or Cromemco of problems encountered during the warranty period. In order to obtain warranty service, first obtain a return authorization number by contacting the Authorized Cromemco Dealer from whom you purchased the product. Then attach to the product:

1. Your name, address and telephone number,
2. the return authorization number,
3. a description of the problem, and
4. proof of the date of retail purchase.

Ship or otherwise return the product, transportation and insurance costs prepaid, to the Authorized Cromemco Dealer. If you are unable to receive warranty repair from the Authorized Cromemco Dealer from whom you purchased the product, you should contact Cromemco Customer Support at: Cromemco, Inc., 280 Bernardo Ave., Mountain View, Ca. 94043.

What Is Not Covered By This Warranty:

Cromemco does not warrant any products, components or parts not manufactured by Cromemco.

This warranty does not apply if the product has been damaged by accident, abuse, misuse, modification or misapplication; by damage during shipment; or by improper service. This product is not warranted to operate satisfactorily with peripherals or products not manufactured by Cromemco. Transportation and insurance charges incurred in transporting the product to and from the Authorized Cromemco Dealer or Cromemco are not covered by this Warranty.

Exclusion of Liability, Damages, and Other Warranties:

THIS WARRANTY IS IN LIEU OF ALL OTHER WARRANTIES, WHETHER ORAL OR WRITTEN, EXPRESS OR IMPLIED. ANY IMPLIED WARRANTIES, INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO NINETY (90) DAYS FROM THE DATE OF PURCHASE OF THIS PRODUCT. IF THIS PRODUCT IS NOT IN GOOD WORKING ORDER AS WARRANTED ABOVE, YOUR SOLE REMEDY SHALL BE REPAIR OR REPLACEMENT AS PROVIDED ABOVE. CROMEMCO SHALL NOT BE LIABLE FOR INCIDENTAL AND/OR CONSEQUENTIAL DAMAGES FOR THE BREACH OF ANY EXPRESS OR IMPLIED WARRANTY, INCLUDING DAMAGE TO PROPERTY AND, TO THE EXTENT PERMITTED BY LAW, DAMAGES FOR PERSONAL INJURY, EVEN IF CROMEMCO HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. THE AGENTS, DEALERS, AND EMPLOYEES OF CROMEMCO ARE NOT AUTHORIZED TO MAKE MODIFICATIONS TO THIS WARRANTY, OR ADDITIONAL WARRANTIES BINDING ON CROMEMCO ABOUT OR FOR PRODUCTS SOLD OR LICENSED BY CROMEMCO. ACCORDINGLY, ADDITIONAL STATEMENTS WHETHER ORAL OR WRITTEN EXCEPT SIGNED WRITTEN STATEMENTS FROM AN OFFICER OF CROMEMCO DO NOT CONSTITUTE WARRANTIES AND SHOULD NOT BE RELIED UPON. SOFTWARE, TECHNICAL INFORMATION AND FIRMWARE ARE LICENSED ONLY BY A SEPARATE AGREEMENT ON AN "AS IS" BASIS.

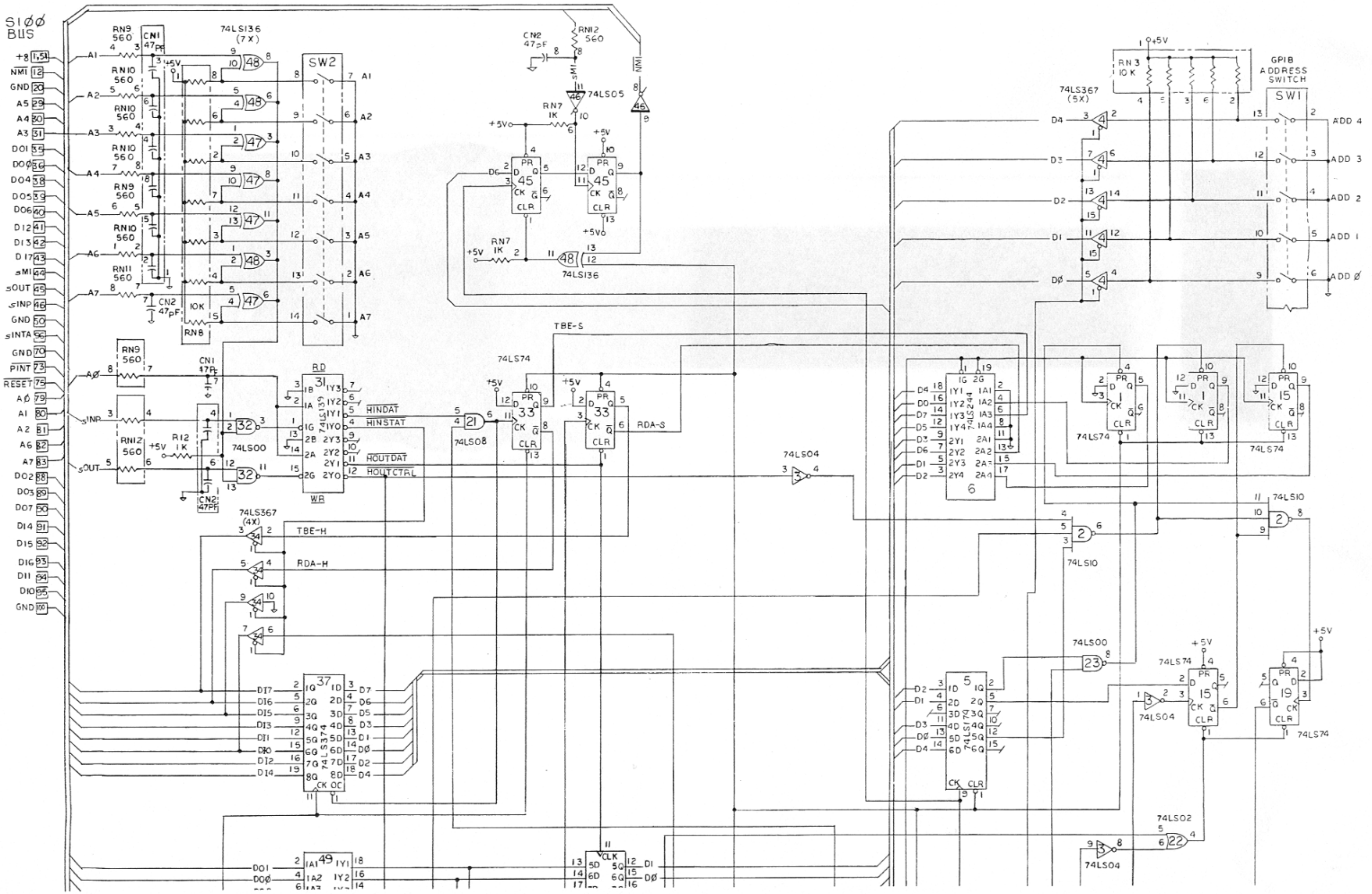
Limitation on Statute of Limitation and Transferability:

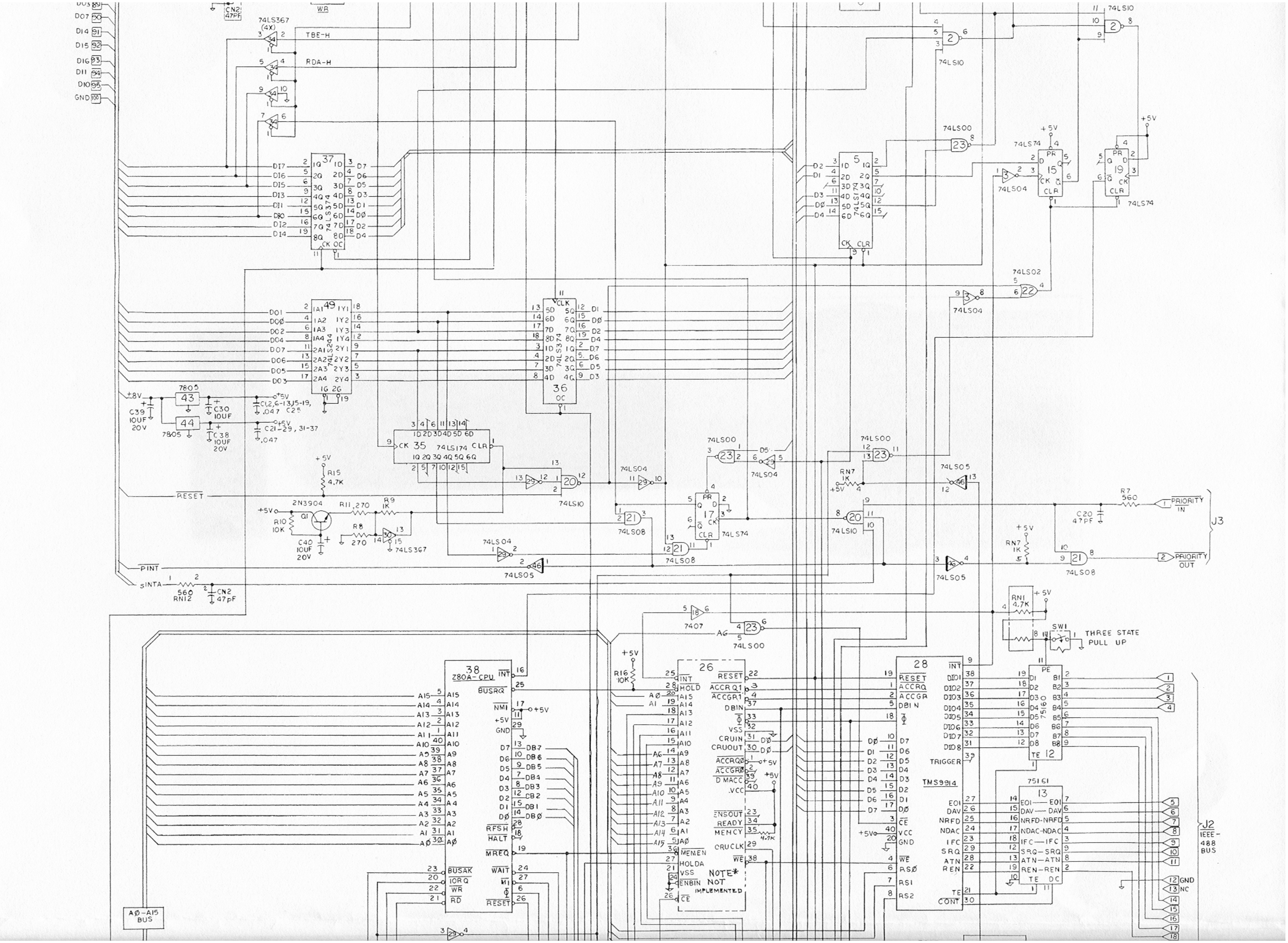
This warranty and the statute of limitations shall run concurrently with any acceptance period. This warranty is not transferable. No suit, litigation, or action shall be brought based on the alleged breach of this warranty or implied warranties more than one year after the date of purchase in those jurisdictions allowing such a limitation, otherwise no such action shall be brought more than one year after the expiration of this warranty.

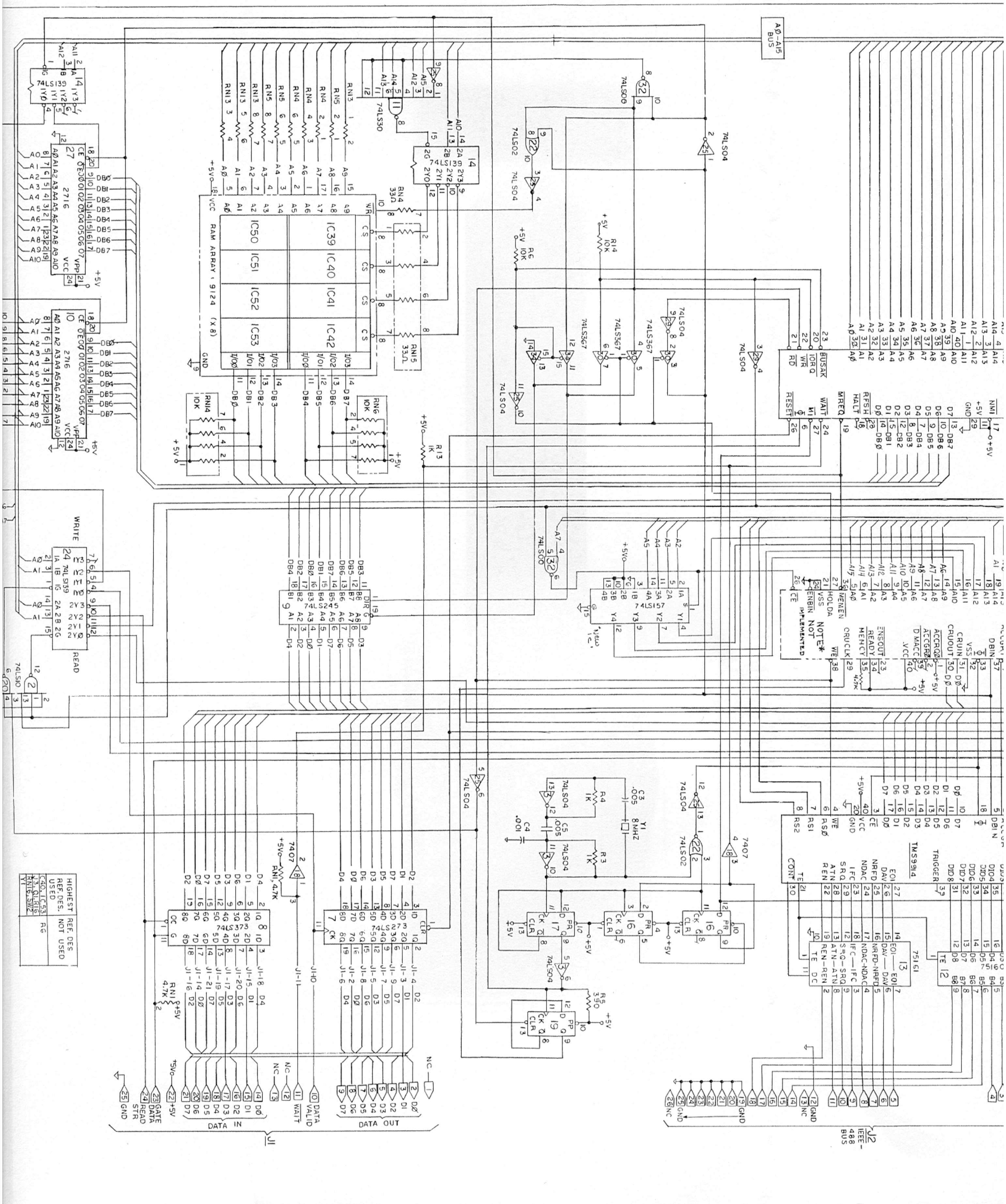
Other Important Provisions:

Some states do not allow the exclusion or limitation of incidental or consequential damages or limitations on how long an implied warranty lasts, so the above limitation or exclusion may not apply to you. This warranty shall not be applicable to the extent that any provision of this warranty is prohibited by any federal, state or municipal law which cannot be preempted. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

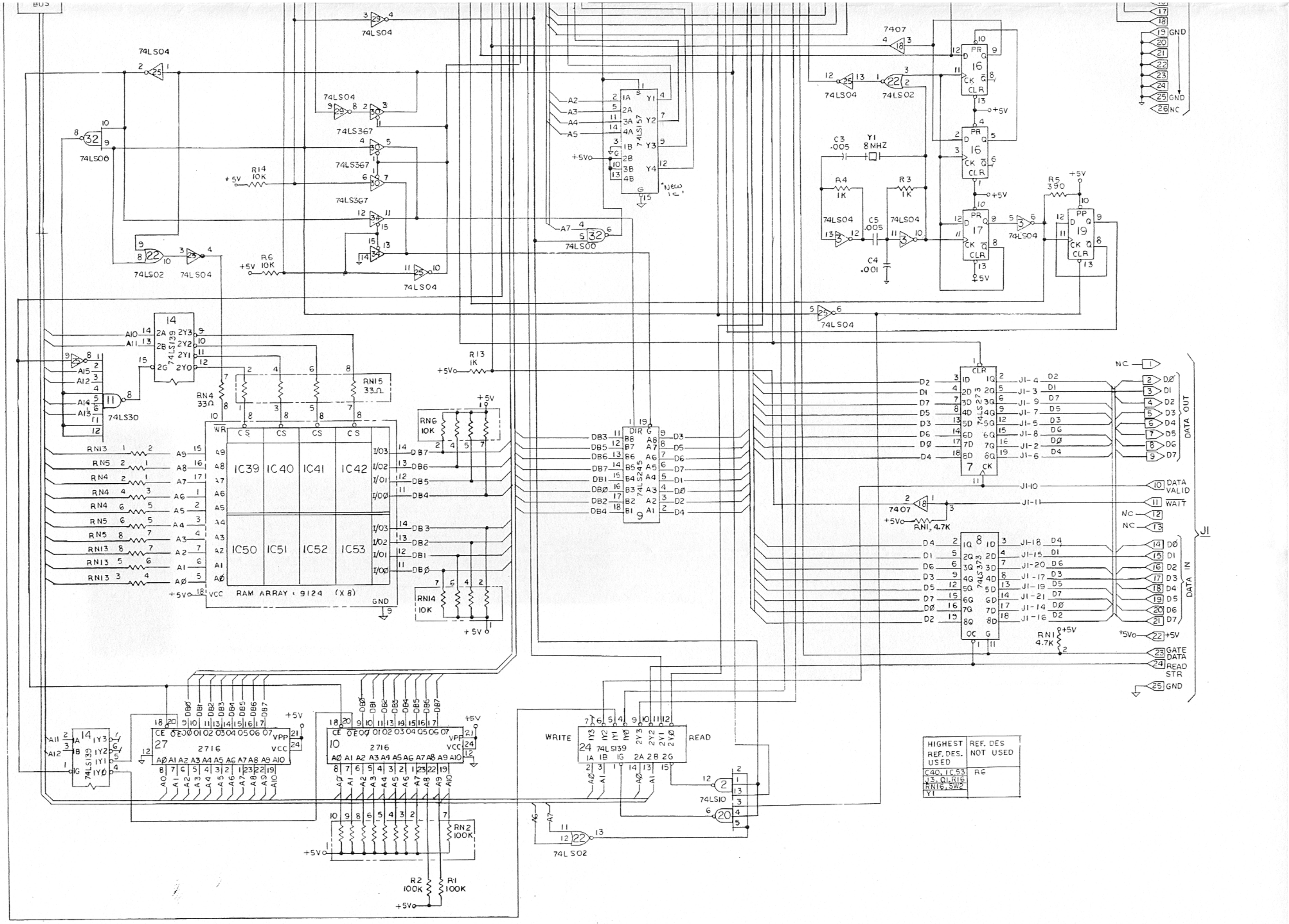
7/82


S100 BUS
74LS136 (7X)
SW2
RN9 560
RN10 560
RN11 560
RN12 560
CN1 47PF
CN2 47PF
RN8 10K
RN7 1K
RN12 1K
74LS00
74LS04
74LS08
74LS74
74LS367 (4X)
74LS367 (5X)
74LS139
74LS244
74LS374
74LS10
74LS02
RD
WR
HINDAT
HINSTAT
HOUTDAT
HOUTCTRL
TBE-H
RDA-H
TBE-S
RDA-S
RN3 10K
GPIB ADDRESS SWITCH
SW1
ADD 4
ADD 3
ADD 2
ADD 1
ADD Ø


AØ-A15 BUS
Z80A-CPU
74LS374
74LS244
74LS174
74LS367
TMS9914
75160
75161
THREE STATE PULL UP
PRIORITY IN
PRIORITY OUT
J3
J2 IEEE-488 BUS
NOTE* NOT IMPLEMENTED



AØ–A15 BUS
RAM ARRAY 9124 (x8)
IC50 IC51 IC52 IC53
IC39 IC40 IC41 IC42
2716
74LS139
74LS245
74LS157
74LS373
74LS273
75161
75160
TMS9914
NOTE* NOT IMPLEMENTED
WRITE
READ
DATA IN
DATA OUT
J1
J2 IEEE-488 BUS
HIGHEST REF. DES. USED
REF. DES. NOT USED



TITLE	SCHEMATIC General Purpose Interface Board
DRAFTSMAN	DUVALL 7/5/81
ID NUMBER	GPIB-040-0053
REV	2
SCALE	None

Cromemco Incorporated