The HPC as a Front-End Processor

National Semiconductor
Application Note 551
Brian Marley
April 1992

ABSTRACT

This application note covers the use of the National Semiconductor HPC46083 High-Performance microController as a front-end processor to collect and block data from RS-232 (serial) and Centronics (parallel) ports for a Host CPU (a typical application being an intelligent graphics-oriented printer). This application note builds on Application Note AN-550 (UPI Port); the result being a program that implements a versatile front-end processor for a National NS32CG16 CPU.

1.0 INTRODUCTION

In Application Note AN-550, ''A Software Driver for the HPC Universal Peripheral Interface Port'', we saw how a National Semiconductor HPC46083 microcontroller can be connected and programmed to perform intelligent peripheral functions for a host CPU; our example being an application connecting an NS32CG16 CPU through the HPC to a typical front panel.

In this application note, we will expand on the hardware and the driver software presented there in order to implement a very useful function for a high-performance microcontroller: that of a front-end processor for data collection. To demonstrate a real-world application for this kind of function, we implement here an intelligent interface to a Centronics-style parallel input port and an RS-232 serial port, typical of a graphics-oriented printer.

2.0 THE FRONT-END PROCESSOR FUNCTION

As systems start to support higher data rates, one of the ever-present challenges is to minimize the interrupt processing load on the CPU, which can become intolerable if the CPU must process each character received in a separate interrupt. Since the character transfer task is typically so simple (reading a character from an input port and placing it into a memory buffer), it is often the case that the unavoidable context switch time associated with the interrupt outweighs the time spent processing the input character. In addition, the communication task may not be the CPU's highest priority: for example, in band-style laser printers the CPU must keep up with the paper movement; it can neither rerun an image nor stop the paper. The communication rate therefore suffers; even printers running from a Centronics-style parallel port are typically unable to accept data faster than 4k characters per second.

The traditional technique for overcoming this obstacle is to implement Direct Memory Access (DMA) for the communication ports. This is, however, quite a large investment in hardware, requiring an external DMA controller chip and more sophisticated bus structures to support it. In other words, it may be acceptable for a computer system, but it is overly expensive for an embedded controller application. Also, the response time required of the CPU can still be stringent, especially in implementing flow control to pace the character rate from the external system presenting the data.

The HPC46083 microcontroller, however, allows a much more cost-effective approach to the problem. As a peripheral, it interfaces to the CPU much as any peripheral controller would. In the application documented here, it buffers up to 128 characters before interrupting the CPU, thus dropping the CPU input interrupt processing frequency by over two orders of magnitude, while allowing a character input rate of over 20 kb/sec.

2.1 Data Transfer Technique

The benefit provided by a front-end processor is derived from the efficiency it adds to the process of getting data into the CPU's data buffer; that is, how much of the CPU's processing time gets dedicated to this task.

The efficiency is provided by two means:

1. Reduction of interrupt overhead. By interrupting the CPU only once every 100 characters, the overhead per character becomes virtually negligible.
2. Elimination of error testing overhead. If the CPU were communicating with a UART directly, it would have to poll for error conditions on each character. In our implementation, there are two interrupt vectors for data transfer: one for good data (which transfers a block of data), and one for bad data (which transfers one character and its error flags). The good data interrupt routine, then, which is invoked almost exclusively, contains a very simple inner loop. After reading the character count from the HPC, all that the CPU needs to do is:

— Move a character from the HPC's OBUF register to the current destination address. No time is wasted polling the HPC status; the hardware synchronization technique described in Application Note AN-550 handles this.

— Increment the destination address. (Checking against buffer limits could be done here, but is more efficiently handled outside the inner loop).

— Decrement the character count and test it; loop if nonzero.

The HPC firmware also supports this technique by guaranteeing that the reporting of character errors (and BREAK conditions) is synchronized with good data, so that the CPU can tell exactly where in the data stream the error occurred.

2.2 Logic Replacement

Front-end processing tasks by no means use up the HPC's capabilities in a system. In our application, the HPC also serves as the CPU's only interrupt controller, allowing a large number of vectors with no additional hardware. It performs additional control tasks such as dynamic RAM refresh request timing, front panel control and real-time clock functions given in Application Note AN-550 with inexpensive interfacing. In a single 4 kbyte program developed in our group, we were also able to add an interface to an inexpensive serial EEPROM device (connected directly to the MICROWIRE/PLUS™ port of the HPC) and to a laser-printer engine for non-imaging control functions, and we also implemented a higher-resolution event timing feature. (These are topics for future application notes, however, and are not dealt with here.)

To summarize, then, the HPC not only can provide front-end processing functions, but can pay for itself by replacing other logic in the system.

MICROWIRE/PLUS™ is a trademark of National Semiconductor Corporation.

RRD-B30M105/Printed in U. S. A.

3.0 HARDWARE

The following sections refer to the schematic pages included. We will discuss here only the portions involving the Centronics Parallel and RS-232 Serial ports. See Application Note AN-550 for details of the other connections shown (the UPI port and front-panel functions).

3.1 The Centronics Parallel Port

The Centronics port was implemented on the connector designated J5. Most of the interface is diagrammed on Sheet 4 of the schematic.

3.1.1 Control Inputs

Pin 1 of the J5 connector receives the Data Strobe ($\overline{\text{STROBE}}$) input, which signals the presence of valid data from the external system. On Sheet 4, in area C5, this signal appears from the connector. It is filtered using a Schmitt trigger (a spare 1488 RS-232 receiver chip), and is then presented to the HPC (Sheet 3) as interrupt signal I4.

Pin 31 is the Input Prime signal ($\overline{\text{PRIME}}$), which is asserted low by the external system in order to reset the interface. It appears on Sheet 4 in area D5, and is filtered in a similar manner. It is then gated with the signal ENPRIME from the Centronics Control Latch, and the resulting signal is presented to the HPC on pin *EXUI, which is the External UART Interrupt input. The gating is used to prevent confusion between UART and $\overline{\text{PRIME}}$ interrupts: while the Centronics port is selected, only $\overline{\text{PRIME}}$ causes interrupts, and while the RS-232 port is selected, this gating keeps $\overline{\text{PRIME}}$ interrupts from being asserted.

3.1.2 Data Inputs

Eight data bits, from J5 pins 2 through 9, appear in areas B8 and C8 of Sheet 4. They are latched into a 74LS374 latch on the leading edge of the $\overline{\text{STROBE}}$ signal (note the inversion through the Schmitt receiver on $\overline{\text{STROBE}}$). The latch is enabled to present data to the HPC's Port D pins by the signal $\overline{\text{ENCDATA}}$, which comes from HPC pin B12. Note that Port D is also used for inputting pushbutton switch data from a front panel.

3.1.3 Control Outputs

The Centronics control and handshake signals are presented by loading the Centronics Control Latch (Sheet 4, area B4) from the HPC's pins A8 through A15 (Port A Upper) using as a strobe the signal CCTLCLK from HPC pin P2.

Pin 10 of connector J5 is the Centronics Acknowledge ($\overline{\text{CACK}}$) pulse, which is used to signal the external system that the HPC is ready for the next byte of data. This is one of the two handshake signals used to pace data flow. It is initialized high by the HPC, and is pulsed low when required.

Pin 11 is the Centronics Busy (CBUSY) signal, which is generated by the flip-flop on Sheet 4, area C3. It is set directly by a $\overline{\text{STROBE}}$ pulse, and is also loaded from the Centronics Control Latch whenever the HPC finishes reading a byte of data (rising edge of $\overline{\text{ENCDATA}}$). This will clear CBUSY under normal conditions, allowing the external system to send another byte of data.

Five additional signals, whose functions vary significantly from printer to printer, are presented on connector J5 from the Centronics Control Latch. These are:

Pin 13, which generally indicates that the printer is selected.

Pin 12, which indicates that the printer needs attention (for example, that it is out of paper).

Pin 32, which indicates a more permanent or unusual problem (lamp check or paper jam).

Pins 33 and 35, which vary more widely in use.

These five pins are manipulated by commands from the CPU; the HPC simply presents them as commanded.

3.1.4 Other Signals

Pin 18 of the Centronics port connector receives a permanent +5V signal (area B2 of Sheet 4), and a set of other pins (middle of Sheet 2) are connected permanently to ground.

3.2 The RS-232 Serial Port

The serial port (on connector J6) makes use of the HPC's on-chip UART and baud rate generator; very little off-chip hardware is required. The entire RS-232 circuit appears on Sheet 3 of the schematic.

This port is implemented in a way typical of printers, and so there are no sophisticated handshaking connections. The interface looks like an RS-232 DTE device: Connector J6 pin 2 is transmitted data (out) and pin 3 is received data (in).

The RS-232 data input appears in area B8 of Sheet 3, as signal RXD. After the RS-232 receiver, it is presented on the HPC's UART input pin (I6). Note that this pin can be monitored directly as a port bit; this enables the HPC to check periodically for the end of a BREAK condition without being subjected to a constant stream of interrupts for null characters.

The Data Set Ready signal (DSR) is received from pin 6 of J6, and presented on HPC pin I7, where it can be monitored by the HPC firmware.

The Request to Send signal (RTS) is a constant high level placed on J6 pin 4.

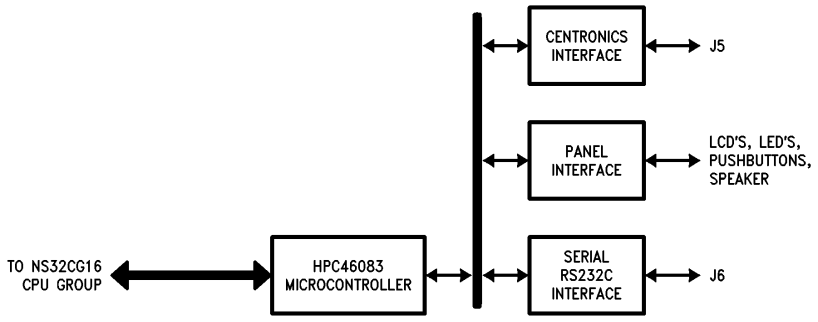
Transmitted data (TXD) is presented from the HPC's UART output pin (B0), through a buffering gate, to an RS-232 driver, and then out on J6 pin 3. The buffering gate would be unnecessary if the CMOS 14C88 driver were being used, but the gate was a spare and allowed cost savings using the less expensive TTL 1488 chip.

Data Terminal Ready (DTR) is simply presented from a programmable port pin of the HPC (pin B1). It is buffered through a spare inverter, and then presented to RS-232 connector J6 pin 20 through an RS-232 driver. As with the UART output, the buffering would be unnecessary with the 14C88 type of RS-232 driver; however, note that the HPC firmware would have to be modified slightly due to the resulting polarity difference on the pin.

J6 pins 1 (Frame Ground) and 7 (Signal Ground) are, of course, grounded, as shown in this sheet also.

3.3 Schematic Sheets

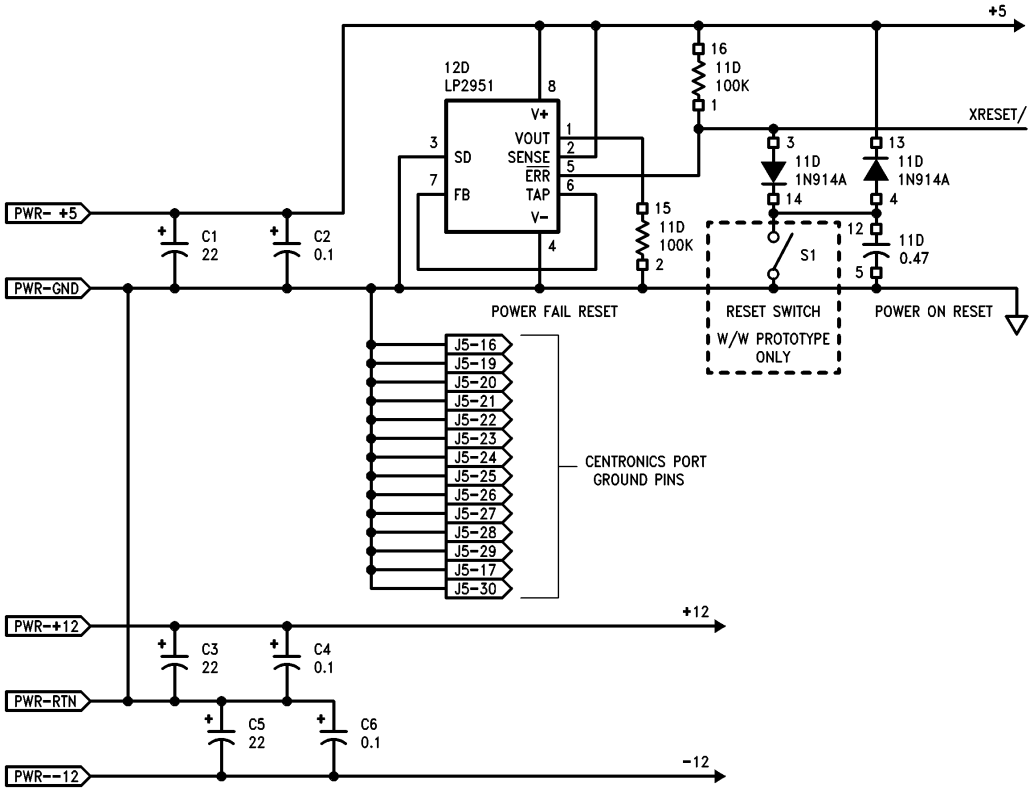
Sheet 1



TL/DD/9977–1

Power and Ground Distribution

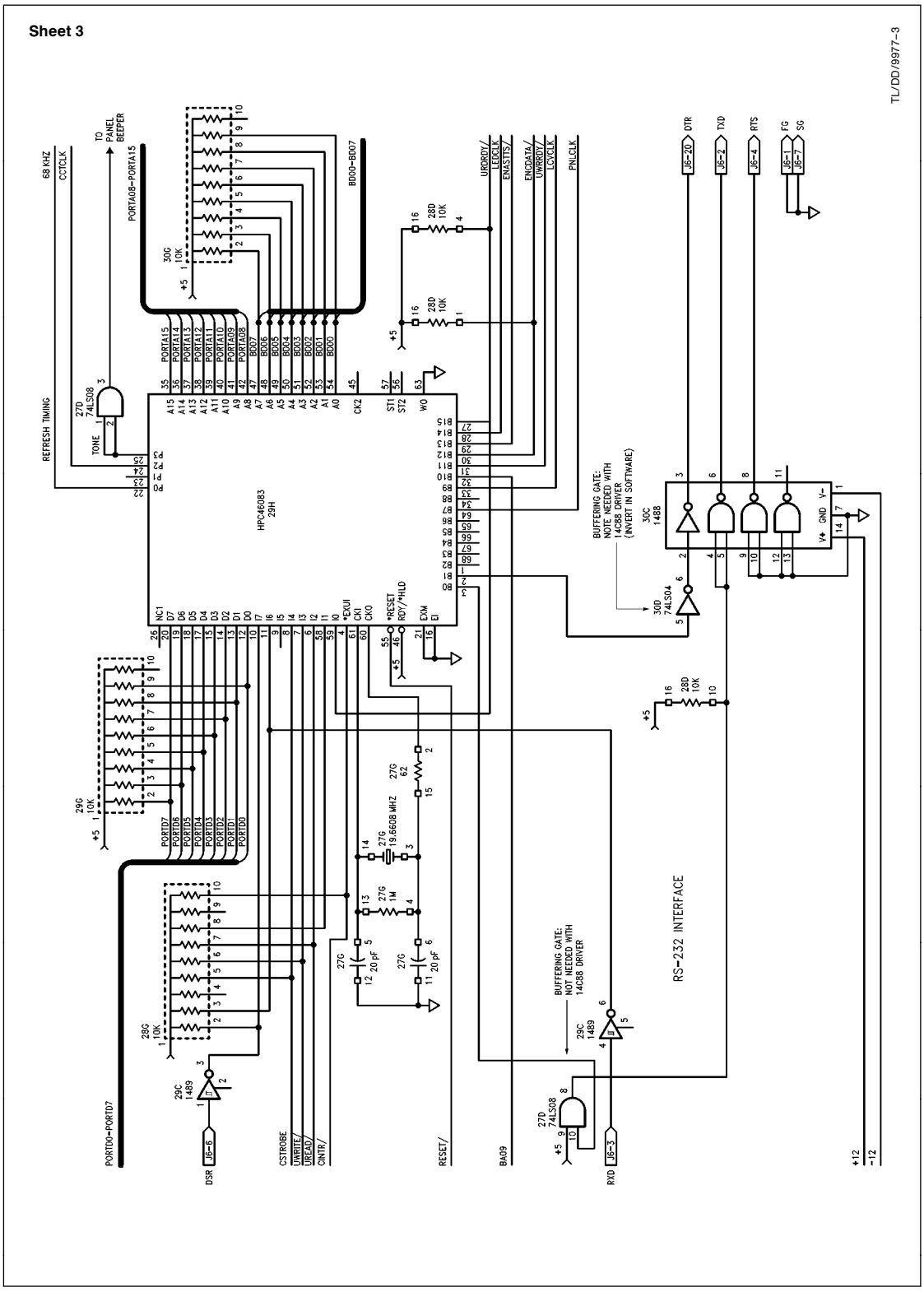
Sheet 2



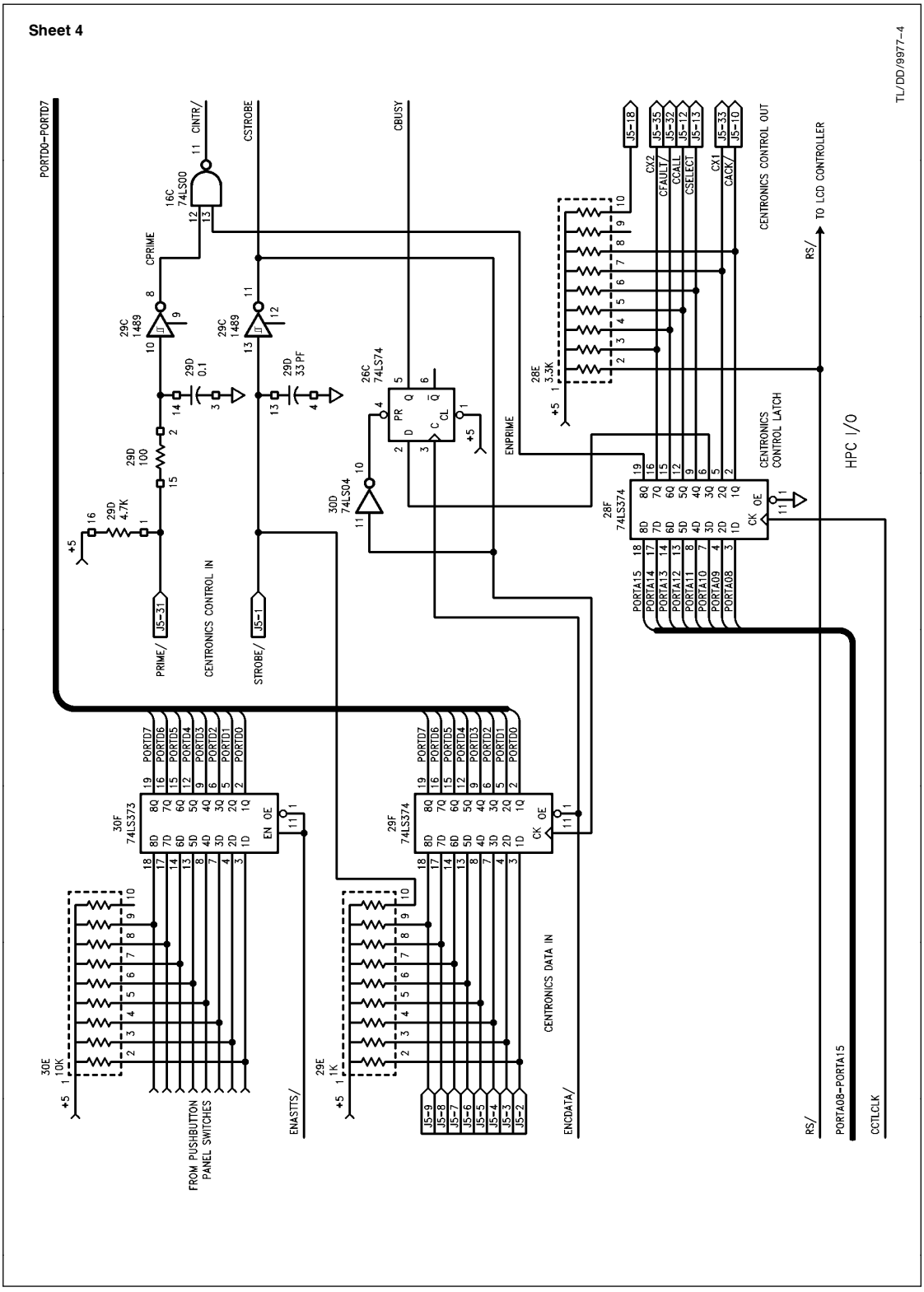
TL/DD/9977–2

Notes: (Unless otherwise specified)

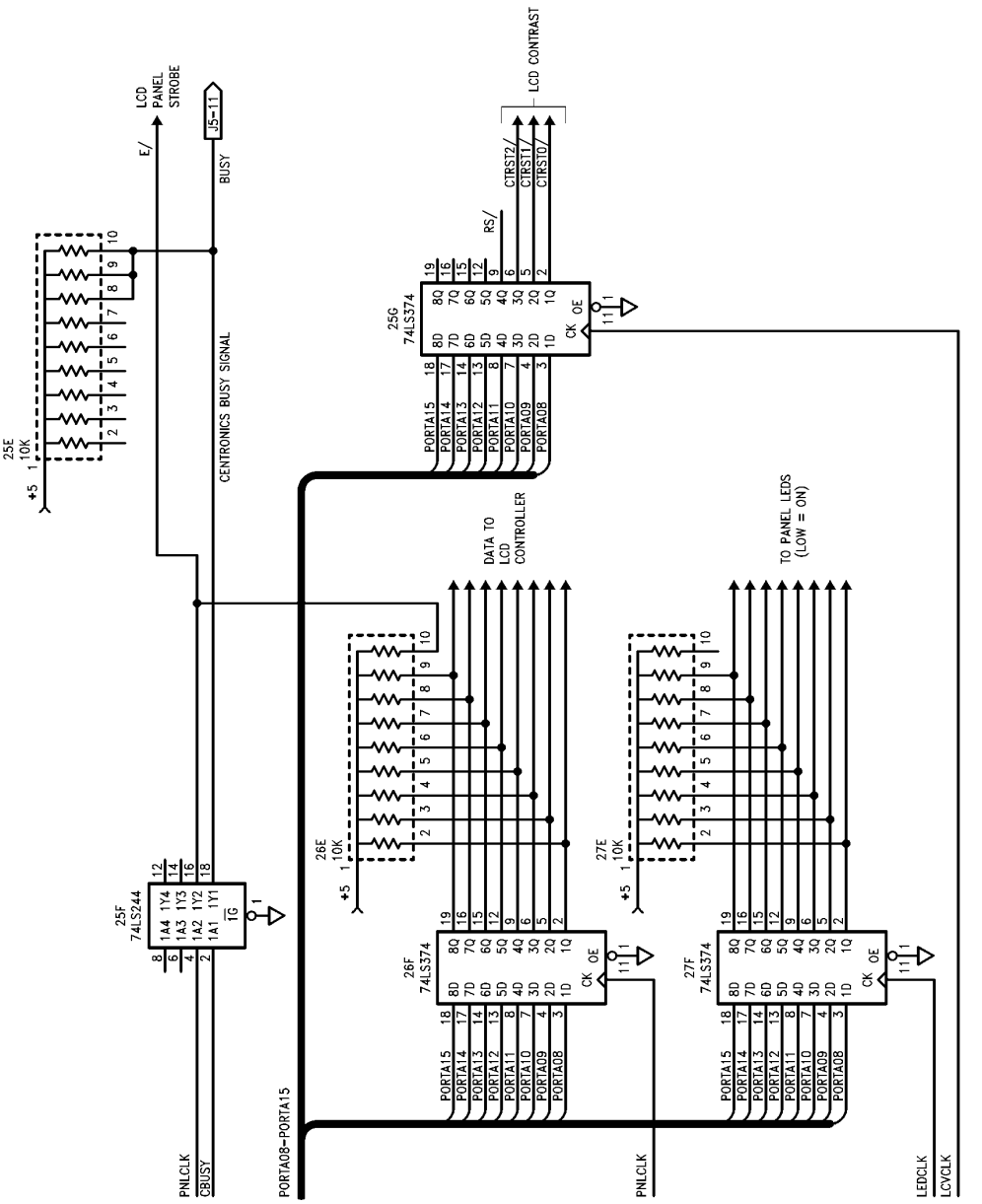
1. All capacitance values in microfards, 50V.
2. All resistor values in Ohms, ¼W, 5%.


Sheet 3
TL/DD/9977–3
HPC46083
29H
RS-232 INTERFACE
BUFFERING GATE:
NOTE NEEDED WITH
14C88 DRIVER
(INVERT IN SOFTWARE)
BUFFERING GATE:
NOT NEEDED WITH
14C88 DRIVER
REFRESH TIMING
68 KHZ
CCTCLK
TO
PANEL
BEEPER
TONE
PORTA08–PORTA15
BD00–BD07
PORTD0–PORTD7
CSTROBE
UWRITE/
UREAD/
CINTR/
RESET/
BA09
DSR
RXD
DTR
TXD
RTS
FG
SG
+12
–12
URDRDY/
LEDCLK
ENASTS/
ENCDATA/
UWRRDY/
LCVCLK
PNLCLK
19.6608 MHZ
1M
20 pF
62
10K
28D
74LS08
74LS04
1489
14C88

Sheet 4


PORTD0–PORTD7
CINTR/
CSTROBE
CBUSY
CPRIME
ENPRIME
PRIME/
STROBE/
CENTRONICS CONTROL IN
CENTRONICS CONTROL OUT
CENTRONICS DATA IN
CENTRONICS CONTROL LATCH
TO LCD CONTROLLER
HPC I/O
FROM PUSHBUTTON PANEL SWITCHES
ENASTTS/
ENCDATA/
RS/
PORTA08–PORTA15
CCTLCLK
CFAULT/
CCALL
CSELECT
CACK/
16C 74LS00
29C 1489
29D 100
29D 4.7K
29D 0.1
29D 33 PF
26C 74LS74
30D 74LS04
28E 3.3K
28F 74LS374
30F 74LS373
29F 74LS374
30E 10K
29E 1K
TL/DD/9977–4

Sheet 5


TL/DD/9977–5
LCD PANEL STROBE
E/
J5–11
BUSY
CENTRONICS BUSY SIGNAL
25E 10K
+5
LCD CONTRAST
CTRST2/
CTRST1/
CTRST0/
RS/
25G 74LS374
PORTA15
PORTA14
PORTA13
PORTA12
PORTA11
PORTA10
PORTA09
PORTA08
DATA TO LCD CONTROLLER
TO PANEL LEDS (LOW = ON)
26E 10K
27E 10K
26F 74LS374
27F 74LS374
25F 74LS244
PORTA08–PORTA15
PNLCLK
CBUSY
LEDCLK
LCVCLK

4.0 PROTOCOL

The command and interrupt protocol is a superset of that implemented for Application Note AN-550. The two commands SELECT-CENT and SELECT-UART are added to select and initialize each of the communication ports (Centronics or RS-232). The CPU can exercise control over data buffering by the commands FLUSH-BUF, CPU-BUSY, CPU-NOT-BUSY and SET-IFC-BUSY. It can set Centronics port error flags and status using SET-CENT-STS, and it can test for RS-232 status using the TEST-UART command. The HPC also allows the CPU to send characters out on the RS-232 port using the SEND-UART command.

New interrupts presented by the HPC are !DATA, which transfers up to 128 bytes of buffered data to the CPU, !PRIME and !UART-STATUS, which inform the CPU of port status changes, and !DATA-ERR, which reports in detail any error ocurring in characters received. The interrupt !ACK-UART is presented to the CPU to acknowledge that the SEND-UART command has been completed.

Note that the command codes for the front panel functions have been changed. Their formats, however, have not changed, nor have their functions, except that the INITIALIZE command now performs a disconnection function on the RS-232 and Centronics ports.

4.1 Commands

The first byte (command code) is sent to address FFFC00, and any argument bytes are then written to address FFFE00. The CPU may poll the UPIC register at address FD0000 to determine when the HPC can receive the next byte, or it can simply attempt to write, in which case it will be held in Wait states until the HPC can receive it. Except where noted, the CPU may send commands continuously without waiting for acknowledgement interrupts from previous commands.

00 INITIALIZE

This command has two functions. The first INITIALIZE command after a hardware reset (or RESET-HPC command) enables the !RTC and !BUTTON-DATA interrupts. Both data communcation ports are set to their ''Busy'' states until a ''SELECT'' command is sent. The INITIALIZE command may be re-issued by the CPU to de-select both communication ports, and to either start or stop the !RTC interrupts. There is one argument:

RTC-Interval: One-byte value. If zero, !RTC interrupts are disabled. Otherwise, the !RTC interrupts occur at the interval specified (in units of 10 ms per count).

01 SELECT-CENT

Select the Centronics port and set it ready, using the timing sequence specified by the supplied ACK-Mode argument. Data from the port is enabled, and the !PRIME interrupt is also enabled. Arguments:

ACK-Mode: one byte in the format:

x	x	x	x	x	L	Timing

where the Timing field is encoded as:

00 = BUSY falling edge occurs after $\overline{ACK}$ pulse.

01 = BUSY falling edge occurs during $\overline{ACK}$ pulse.

10 = BUSY falling edge occurs before $\overline{ACK}$ pulse.

and the L bit, when set, requests Line Mode. It suppresses the removal of BUSY and the occurrence of the $\overline{ACK}$ pulse when the buffer is passed to the CPU. To fully implement Line Mode, this mode should be used with Pass-Count = 1 and Stop-Count = 1, and the CPU must use the SET-CENT-STS command to acknowledge each character itself.

Pass-Count: Number of characters in buffer before the HPC passes them automatically to CPU. One byte.

Stop-Count: Number of characters in buffer before HPC tells the external system to stop. One byte.

Note that the buffer is a maximum of 128 bytes in length, in this implementation.

Requires INITIALIZE command first.

02 SELECT-UART

Select Serial port and set it ready, according to supplied arguments. Requires INITIALIZE command first. Arguments are:

Baud: Baud rate selection. One Byte containing.

0 = 300 baud

1 = 600 baud

2 = 1200 baud

3 = 2400 baud

4 = 4800 baud

5 = 9600 baud

6 = 19200 baud

7 = 38400 baud

8 = 76800 baud

Frame: One byte, selecting character length, parity and number of stop bits.

Value	Data Bits	Parity	Stop Bits
0	8	Odd	1
1	8	Even	1
2	8	None	1
3	8	None	2
4	7	Odd	1
5	7	Even	1
6	7	Odd	2
7	7	Even	2

Flow: One byte, bit-encoded for handshaking and flow control modes:

7	6	5	4	3	2	1	0
0	0	0	0	XON	DTR		DSR

DSR: 1 = the HPC disables the UART receiver while the DSR input is inactive.

DTR: Polarity of DTR output, and whether it is used as a flow-control handshake.

00 = Permanently low (negative voltage).

01 = Permanently high (positive voltage).

10 = Handshaking: low means ready.

11 = Handshaking: high means ready.

XON: 1 = the HPC performs XON/XOFF flow control.

Pass-Count: Number of characters in buffer before the HPC passes them automatically to CPU. One byte.

Stop-Count: Number of characters in buffer before HPC tells the external system to stop. One byte.

Note that the buffer is a maximum of 128 bytes in length, in this implementation.

Requires INITIALIZE command first.

03 (reserved)

04 FLUSH-BUF — No arguments. Flush HPC data communication buffer to CPU. Any data in the buffer is immediately sent to the CPU (using the !DATA interrupt). This command triggers the !DATA interrupt only if the buffer contains at least one byte. Requires INITIALIZE command and SELECT command first.

05 CPU-BUSY — No arguments. Indicates that the CPU cannot accept any more data (the CPU's data buffer is full). This suppresses the !DATA and !DATA-ERR interrupts. Requires INITIALIZE command and SELECT command first.

06 CPU-NOT-BUSY — No arguments. This undoes a previous CPU-BUSY command, and indicates that the CPU can now accept more data from the HPC. Requires INITIALIZE command and SELECT command first.

07 SET-IFC-BUSY — ''Set Interface Busy''. No arguments. Commands the HPC to immediately signal the external system to stop sending characters. This status is removed only by performing a SELECT command. Requires INITIALIZE command and SELECT command first.

08 SET-CENT-STS — ''Set Centronics Port Status''. Loads Centronics latch from the supplied argument byte. Argument is eight bits, which must be encoded as follows:

ENPRIME	CX2	$\overline{\text{FAULT}}$	CALL	SELECT	BUSY	CX1	$\overline{\text{ACK}}$

The $\overline{\text{ACK}}$ bit should always be a ''1''. The CPU must use the BUSY bit to generate an $\overline{\text{ACK}}$ pulse: if the BUSY bit is zero, the $\overline{\text{ACK}}$ signal will be automatically pulsed low, then high, (regardless of the previous states of BUSY and $\overline{\text{ACK}}$).

Requires INITIALIZE command and SELECT-CENT command first.

09 SET-CONTRAST — The single argument is a 3-bit number specifying a contrast level for the LCD panel (0 is least contrast, 7 is highest contrast). There is no response interrupt. Does not require INITIALIZE command first.

0A SEND-LCD — This writes a string of up to 8 bytes to the LCD panel. Arguments are:

flags: A single byte, containing the RS bit associated with each byte of data. The first byte's RS value is in the least-significant bit of the FLAGS byte.

#bytes: The number of bytes to be written to the LCD display.

byte[1]–byte[#bytes]: The data bytes themselves.

The HPC determines the proper delay timing required for command bytes (RS = 0) from their encodings. This is either 4.9 ms or 120 μs.

The response from the HPC is the !ACK-SEND-LCD interrupt, and this command must not be repeated until the interrupt is received. This command does not require an INITIALIZE command first.

0B SEND-LED — The singe argument is a byte containing a ''1'' in each position for which an LED should be lit.

There is no response interrupt, and this command does not require the INITIALIZE command first.

0C BEEP — No arguments. This beeps the panel for approximately one second. No response interrupt. If a new BEEP command is issued during the beep, no error occurs (the buzzer tone is extended to one second beyond the most recent command). Does not require INITIALIZE command first.

0D SEND-UART	The single one-byte argument is sent on the UART port. An acknowledgement interrupt !ACK-UART occurs on completion. This command must not be repeated until the interrupt is received. Requires INITIALIZE and SELECT-UART commands first.
0E TEST-UART	Triggers a !UART-STATUS interrupt. This command must not be repeated until the interrupt is received. No arguments. Requires INITIALIZE and SELECT-UART commands first.
A5 RESET-HPC	Resets the HPC if it is written to address FFFC00. It may be written at any time that the UPI port is ready for input; it will automatically cancel any partially-entered command. The CPU's Maskable Interrupt must be disabled before issuing this command. After issuing this command, the CPU should first poll the UPIC register at address FD0000 to see that the HPC has input the command (the least-significant bit [Write Ready] is zero). It must then wait for at least 25 μs, then read a byte from address FFFE00. The HPC now begins its internal re-initialization. The CPU must wait for at least 80 μs to allow the HPC to re-initialize the UPI port. Since part of the RESET procedure causes Ports A and B to float briefly (this includes the CPU's Maskable Interrupt input pin), the CPU should keep its maskable interrupt disabled during this time. It also must not enter a command byte during this time because the byte may be lost.

4.2 Interrupts

The HPC interrupts the CPU, and provides the following values as the interrupt vectors for the CPU hardware. The CPU then reads data from the HPC at address FFFE00. All data provided by the HPC must be read by the CPU before returning from the interrupt service routine, otherwise the HPC would either hang or generate a false interrupt. The CPU may poll the UPIC register at address FD0000 to determine when each data byte is ready, or it may simply attempt to read from address FFFE00, and it will be held in Wait states until the data is provided by the HPC.

Note: All CPU interrupt service routines, including the NMI interrupt routines, must return using the ''RETT 0'' instruction. Do NOT use ''RETI''.

Vector	
00–0F (none)	(Reserved for CPU internal traps and the NMI interrupt.)
10 !DATA	Buffer data is being transferred to CPU. This will happen either automatically, at a point defined by the most recent SELECT command, or as the result of a FLUSH-BUF command. It is followed by a one-byte Length (number of characters: current HPC firmware has a range of 1–128), then that number of characters. Enabled by SELECT command after at least one INITIALIZE command.
11 !RTC	Real-Time Clock Interrupt. No data returned. Enabled by INITIALIZE command if interval value supplied is non-zero. **Note:** This version of HPC firmware issues a non-fatal !DIAG interrupt if the CPU fails to service each !RTC interrupt before the next one becomes pending.
12 (reserved)	
13 !PRIME	Centronics INPUT PRIME signal has become active. No data returned. Enabled by SELECT-CENT command after at least one INITIALIZE command.
14 (reserved)	
15 (reserved)	
16 (reserved)	
17 !ACK-SEND-LCD	This is the response to the SEND-LCD command, to acknowledge that data has all been written to Panel LCD display. No other data is provided with this interrupt. Always enabled, but occurs only in response to a SEND-LCD command.
18 !BUTTON-DATA	Pushbutton status has changed: one or more buttons have been either pressed or released. The new status of the switches is reported in a data byte, encoded as follows: Any pushbutton that is depressed is presented as a ''1''. All other bit positions, including unused positions, are zeroes. The pushbuttons are debounced before being reported to the CPU. This interrupt is enabled by the first INITIALIZE command after a reset.

19 !UART-STATUS

UART status has changed. This interrupt occurs only while the UART is selected. A data byte shows the UART's new state:

Bit	Condition
0 (LSB)	New state of DSR signal. This causes an interrupt only if DSR monitoring was requested in the last SELECT-UART command. The UART receiver is automatically enabled and disabled by the HPC, so no CPU action is required on receiving this interrupt. If a SELECT-UART command is entered, requesting DSR monitoring, and DSR is inactive, a !UART-STATUS interrupt occurs immediately.
1	This bit is set if a UART BREAK has just ended.
2–7	(unused)

Note 1: If the CPU has issued a CPU-NOT-READY command, this BREAK interrupt may be seen before the !DATA-ERR interrupt that announces the start of the BREAK (and its position in the data stream).

Note 2: The DSR and UART input (BREAK) signals are sampled every 10 ms.

1A !DATA-ERR

An error has been encountered in data coming from the currently-selected communication port. It is enabled by the first SELECT command after the first INITIALIZE command. Two data bytes are returned:

errchr: One byte containing the character on which the error was seen (this character is NOT placed in the data buffer).

errfgs: Error flags, detailing the error seen:

Bit	Error Seen
0 (LSB)	(unassigned)
1	(unassigned)
2	UART BREAK condition detected. This may be preceded by one or two framing errors.
3	**Error Overflow:** More errors occurred than HPC could report (the HPC has no FIFO for error reporting).
4	**Buffer Overflow:** Flow control failed to stop the external system, and the buffer overflowed.
5	**Parity Error:** Serial Port only.
6	**Framing Error:** Serial Port only.
7 (MSB)	**Data Overrun:** Serial Port only.

If bit 2, 3 or 4 is set, the communication port has been automatically shut down by the HPC. The CPU must issue a new SELECT command to re-enable the port.

When a character is received with an error, all characters appearing before it in the buffer are automatically flushed before this interrupt occurs. This is done to preserve the error character's position in the data stream. If the CPU decides to ignore the presence of an error, the character may be simply appended by the CPU to the data already in its data buffer. Please note: If the CPU has issued a CPU-NOT-READY command, the flush cannot occur, and this interrupt will not be issued until the flush has occurred.

1B !ACK-UART

A CPU character has been sent on the UART, and the UART is ready for another. No data is returned with this interrupt. It is always enabled, but occurs only in response to the SEND-UART command.

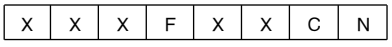
1C (reserved)

1D !DIAG

Diagnostic Interrupt. This interrupt is used to report failure conditions and CPU command errors. There are five data bytes passed by this interrupt:

Severity
Error Code
Data in Error (passed, but contents not defined)
Current Command (passed, but contents not defined)
Command Status (passed, but contents not defined)

The Severity byte contains one bit for each severity level, as follows:

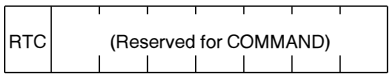


N (Note): least severe. The CPU missed an event; currently only the !RTC interrupt will cause this.

C (Command): medium severity. Not currently implemented. Any command error is now treated as a FATAL error (below).

F (Fatal): highest severity. The HPC has recognized a non-recoverable error. It must be reset before the CPU may re-enable its Maskable Interrupt. In this case, the remaining data bytes may be read by the CPU, but they will all contain the value 1D (hexadecimal). The CPU must issue a RESET command, or wait for a hardware reset. See below for the procedure for FATAL error recovery.

The Error Code byte contains, for non-FATAL errors, a more specific indication of the error condition:



RTC = Real-Time Clock overrun: CPU did not acknowledge the RTC interrupt before two had occurred.

The other bits are reserved for details of Command errors, and are not implemented at this time.

The remaining 3 bytes are not yet defined, but are intended to provide details of the HPC's status when an illegal command is received.

Note: Except in the FATAL case, all 5 bytes provided by the HPC *must* be read by the CPU, regardless of the specific cause of the error.

Fatal Error Recovery:

When the HPC signals a !DIAG error with FATAL severity, the CPU may use the following procedure to recover:

1. Write the RESET command (A5 hex) to the HPC at address FFFC00.
2. By inspecting the UPIC register at address FD0000, wait for the HPC to read the command (the $\overline{\text{WRRDY}}$ bit will go low).
3. Wait an additional 25 μs.
4. Read from address FFFE00. This will clear the OBUF register and reset the Read Ready status of the UPI port. The HPC will guarantee that a byte of data is present; it is not necessary to poll the UPIC register. This step is necessary because only a hardware reset will clear the Read Ready indication otherwise (HPC firmware cannot clear it).
5. Wait at least 80 μs. This gives the HPC enough time to re-initialize the UPI port.
6. After Step 5 has been completed, the CPU may re-enable the Maskable Interrupt and start issuing commands. Since the HPC is still performing initialization, however, the first command may sit in the UPI IBUF register or a few milliseconds before the HPC starts to process it.

5.0 SOURCE LISTINGS AND COMMENTARY

5.1 HPC Firmware Guide

This section is intended to provide help in following the flow of the HPC firmware. Discussion of features already documented in Application Note AN-550 are abbreviated here; see that application note for details.

The firmware for the HPC is almost completely interrupt-driven. The main program's role is to poll mailboxes that are maintained by the interrupt service routines, and to send an interrupt to the CPU whenever a HPC interrupt routine requests one in its mailbox.

On reset, the HPC firmware begins at the label ''start''. However, the first routine appearing in ROM is the Fatal Error routine. This is done for ease of breakpointing, to keep this routine at a constant address as changes are made elsewhere in the firmware.

5.1.1 Fatal Error Routine

At the beginning of the ROM is a routine (label ''hangup'') that is called when a fatal error is detected by the HPC. This routine is identical to that documented in Application Note AN-550.

5.1.2 Initialization

At label ''start'', entered on a Reset signal or by the RESET-HPC command from the CPU, the HPC begins its internal initialization. It loads the PSW register (to select 1 Wait state), and then (at label ''srfsh''), it starts the Refresh clock pulses running for the dynamic RAM by initializing Timer T4 and starting it.

At ''supi'', the UPI port is initialized for transfers between the HPC and the CPU.

At label ''sram'', all RAM within the HPC is initialized to zero.

At ''sskint'', the stack pointer is initialized to point to the upper bank of on-chip RAM (at address 01C0). The address of the fatal error routine ''hangup'' is then pushed, so that it will be called if the stack underflows.

At ''tminit'', the timers T1–T3 are stopped and any interrupts pending from timers T0–T3 are cleared. This step arbitrarily initializes the UART baud rate to 9600, but this selection has no effect.

At ''scent'', the Centronics port is initialized and set up to appear busy to the external system.

At ''suart'', the HPC UART is initialized for serial data from the external system. The RS-232 DTR signal is arbitrarily set low, which generally means that the printer is not ready. The state of DTR is not actually valid until the first SELECT-UART command is received, which selects the handshaking mode.

At ''sled'', the LED control signals are initialized,and all LED indicators are turned off.

At ''stmrs'', all timers are loaded with their initial values, and timers T5–T7 are stopped and any interrupts pending from them are cleared.

At ''slcd'', the LCD display is initialized to a default contrast level of 5, then commands are sent to initialize it to 8-bit, 2-line mode, with the cursor visible and moving to the right by default. This section calls a subroutine ''wrpnl'' for each character; the subroutine simply writes the character in the accumulator out to the LCD display and waits for approximately 10 ms.

The program then continues to label ''minit'', which initializes the variables in the HPC's on-chip RAM to their proper contents.

At label ''runsys'', the necessary interrupts are enabled (from the timers, and from pin I3, which is the UPI port interrupt from the CPU), and the program exits to the Main Program at label ''mainlp''. Interrupts from the Centronics and UART ports are not enabled until the appropriate SELECT command is received.

5.1.3 Main Program (UPI Port Output to CPU)

The Main Program is the portion of the HPC firmware that runs with interrupts enabled. It consists of a scanning loop at label ''mainlp'' and a set of subroutines (explained below). It is responsible for interrupting the CPU and passing data to it; the HPC is allowed to write data to the CPU only after interrupting it. Unlike the simpler UPI/Front Panel interface described in Application Note AN-550, this main loop scans two separate variables in on-chip RAM that are set up by interrupt service routines: a word called ''alert'', and a byte called ''bstat'' (for ''Buffer Status''). Both variables are used to determine whether any conditions exist that should cause an interrupt to the CPU.

The ''alert'' word contains one bit for each interrupt that the HPC can generate. If a bit is set (by an interrupt service routine), the Main Program jumps to an appropriate subroutine to notify the CPU. The subroutine checks whether the UPI interface's OBUF register is empty, and if not, it waits (by calling the subroutine ''rdwait''). It then writes the vector number to the OBUF register. This has the effect of interrupting the CPU (because the pin $\overline{\text{URDRDY}}$ goes low), and the CPU hardware reads the vector from the OBUF register.

If there is more information to give to the CPU, the HPC places it, one byte at a time, into the OBUF register, waiting each time for OBUF to be emptied by the CPU. This technique assumes that the CPU remains in the interrupt service routine until all data has been transferred: if the CPU were to return from interrupt service too early, the next byte of data given to it would cause another interrupt, with an incorrect vector.

(Note, however, that the CPU may be interrupted with a Non-Maskable interrupt from a separate source. This simply inserts a pause into the process of reading data from the HPC. Since the HPC is running its main program at this point, with interrupts still enabled, it will not lose data from its communication port under these circumstances.)

The ''bstat'' byte represents a special case involving the interrupt !DATA to the CPU. This byte shows the main program whether the data communication buffer (which holds data from the external system) is full enough to send its contents to the CPU. If so, the main program calls the subroutine ''snddta'', which interrupts the CPU, then sends one data byte containing the number of characters to be transferred (currently as many as 128 are possible), and then the characters themselves.

The CPU may, at any time, demand that the HPC transfer all characters that are within its communication buffer. (This is called a ''flush'' command, which sets one of the bits of the ''alert'' word, described above.) The HPC, in response, will empty the buffer to the CPU with a !DATA interrupt, even if only one character is left. If the buffer is completely empty, however, the flush command is ignored.

Subroutines called from the Main Program loop are:

sndrtc: sends a Real-Time Clock interrupt to the CPU. No data is transferred; only the interrupt vector.

sndlak: interrupts the CPU to acknowledge that a string of data (from a SEND-LCD command) has been written to the LCD display. No data is transferred for this interrupt.

sndbtn: interrupts the CPU to inform it that a pushbutton has been pressed or released. A data byte is transferred from variable ''swlsnt'', which shows the new states of all the pushbuttons.

sndfsh: performs a Flush operation. If there is data, it jumps to the ''snddta'' routine to send the contents of the buffer to the CPU. If there is no data, however, this subroutine simply returns without generating an interrupt.

snddta: sends data from the communication buffer to the CPU. It may be entered for one of three reasons:

1. the communication buffer is full enough that it must be sent automatically to the CPU.
2. a Flush command has been received from the CPU. (The bit ''aflush'' in the ALERT word is set.)
3. an error has been detected on a character received from the external system. This causes an internal Flush request, so that all good characters are sent to the CPU before the bad character is reported. This case is also different because it does not flush the entire buffer, but only up to the point of the error. The limit is held in the variable ''fshlim''.

The subroutine sends a ''length'' byte (from variable ''numout'', sampled from ''numchr'', which is maintained by the communication interrupt routines). This indicates how many characters will be transferred. The subroutine next sends the characters themselves. It then updates the buffer status variables in on-chip RAM, to indicate how many characters were removed.

Depending on other status of the selected communication port, this subroutine may re-enable communication on the port if it was stopped (for example, if the buffer was too full to accept more data until the ''snddta'' routine emptied it). This is done at label ''sdstp''.

sndprm: interrupts the CPU because the INPUT PRIME signal on the Centronics parallel port was activated by the external system. No data is transferred by this interrupt.

sndust: interrupts the CPU to report a change in UART status. This interrupt may also be triggered by the CPU using the TEST-UART command.

snderr: interrupts the CPU to inform it that a character with an error was received. The character and a byte containing error flags are transferred to the CPU.

snduak: interrupts the CPU in response to a SEND-UART command, to acknowledge that the requested character has been sent on the UART transmitter, and that it is ready to transmit another character.

sndiag: interrupts the CPU to inform it of a !DIAG interrupt condition, when it is of NOTE severity. (Other !DIAG conditions are handled at label ''hangup''.)

5.1.4 UPI Port Input from CPU (Interrupt I3)

This interrupt service routine, at label ''upiwr'', accepts commands from the CPU. Apart from the existence of additional commands, the structure of this routine is identical to that of Application Note AN-550. We document here the labels and functions involved in this larger application.

Command Processing Routines

INITIALIZE	I3 interrupt labels:	State 1 = fcinit	State 3 = lcinit
SELECT-CENT	I3 interrupt labels:	State 1 = fcselc	State 3 = lcselc
SELECT-UART	I3 interrupt labels:	State 1 = fcselu	State 3 = lcselu
FLUSH-BUF	I3 interrupt labels:	State 1 = fcflsh	State 3 = (none)

At label ''fcflsh'', the ''alert'' word bit ''aflush'' is set, which requests that the main program flush the communication buffer.

CPU-BUSY	I3 interrupt labels:	State 1 = fccbsy	State 3 = (none)

At label ''fccbsy'', the buffer status byte ''bstat'' is set to indicate that the CPU is busy and cannot accept more data from the HPC. This disables the !DATA interrupt.

CPU-NOT-BUSY	I3 interrupt labels:	State 1 = fccnby	State 3 = (none)

At label ''fccnby'', the buffer status byte ''bstat'' is set to indicate that the CPU is ready to accept more data from the HPC. The !DATA interrupt is re-enabled.

SET-IFC-BUSY	I3 interrupt labels:	State 1 = fcifby	State 3 = (none)

At label ''fcifby'', the currently selected interface is set busy, in order to present an error indication.

SET-CENT-STS	I3 interrupt labels:	State 1 = fcscst	State 3 = lcscst

At label ''lcscst'', the Centronics Port status byte ''cps'' is loaded from the value supplied by the CPU, and the Centronics port control signals are updated to reflect these new settings. The subroutine ''setcen'' is used to set up the control signals, and it also pulses the Centronics $\overline{\text{ACK}}$ signal if appropriate.

SET-CONTRAST	I3 interrupt labels:	State 1 = fcslcv	State 3 = lcslcv

At label ''lcslcv'' (Set LCD Voltage), the LCD Contrast latch is loaded from the value supplied by the CPU.

SEND-LCD	I3 interrupt labels:	State 1 = fcslcd	State 3 = lcslcd

This command sends a string of up to eight bytes to the LCD display. Application Note AN-550 describes the implementation of this command in detail.

SEND-LED	I3 interrupt labels:	State 1 = fcsled	State 3 = lcsled

At label ''lcslcd'', the byte provided by the CPU is written to the LED latch.

BEEP	I3 interrupt labels:	State 1 = fcbeep	State 3 = (none)

This command sends a one-second beep tone to a speaker.

SEND-UART	I3 interrupt labels:	State 1 = fcsndu	State 3 = lcsndu

At label ''lcsndu'', the single argument (the character to be sent) is placed in variable ''uschr'', and the bit ''schr'' is set in variable ''ups'' (UART Port Status). By doing this, the character has been queued for transmission. The transmission is performed by the subroutine at label ''setuar'', which is also responsible for performing the XON/XOFF flow control protocol. If a character is already being sent (the transmitter interrupt is enabled), then this is the only action required, since the transmitter interrupt automatically invokes the ''setuar'' subroutine. However, if the transmitter is idle, this routine must itself call ''setuar'' to transmit the character.

The subroutine ''setuar'' itself calls another subroutine at label ''uecsnd'', which formats the character to be transmitted into the frame selected by the current UART framing mode. It then sends the character. Note that the UART framing mode applies to output as well as input characters.

TEST-UART	I3 interrupt labels:	State 1 = fcusts	State 3 = (none)

At label ''fcusts'', the HPC sets the ''austat'' bit of the ALERT word, requesting the Main Program to send a !UART-STATUS interrupt to the CPU.

5.1.5 Centronics Commmunication

This task is triggered by each edge of the Centronics port $\overline{\text{STROBE}}$ signal. This signal is detected by the HPC on the I4 interrupt line. On the leading edge of $\overline{\text{STROBE}}$, the character is input to the data communication buffer. This edge also sets the BUSY signal, by hardware action. On the trailing edge, the BUSY flag is affected by the HPC firmware. If the HPC is ready to receive more characters, the BUSY signal is cleared and the $\overline{\text{ACK}}$ signal is pulsed. If the HPC is not ready to receive more data, it leaves the BUSY signal high, which prevents the external system from sending more characters.

The Centronics port $\overline{\text{STROBE}}$ handler is at label ''cenint''. It first determines whether a falling or rising edge was detected on the $\overline{\text{STROBE}}$ signal. If the leading (falling) edge was detected, then it jumps to label ''cstrbl''; otherwise it jumps to label ''cstrbt'' to process a trailing edge.

At label ''cstrbl'', the character is placed in the next available position of the communication buffer, if the buffer is not already full. (If it is already full, then it is processed as an error, as discussed below.) Then some tests are performed:

If the buffer is not full enough to pass data to the CPU, then the routine exits by jumping to label ''cenlex'', where it prepares to detect the trailing edge of $\overline{\text{STROBE}}$. Otherwise, it sets the ''pass'' bit in the variable ''bstat'', which requests the main program to send data to the CPU, and then it continues.

If the buffer is not full enough to tell the external system to stop sending characters, then the routine exits by jumping to ''cenlex''. Otherwise, it sets the ''stop'' bit in variable ''bstat'', indicating that the external system has been stopped, and it also sets the ''cbusy'' flag in variable ''cps'', which will prevent the Centronics BUSY and $\overline{\text{ACK}}$ signals from being changed when the $\overline{\text{STROBE}}$ pulse ends. The routine continues.

If the buffer has become completely full, then the ''full'' bit in ''bstat'' is set, indicating that any more characters received will trigger an error. Character processing then continues at label ''cenlex''.

At ''cenlex'', the Centronics Control Latch is set (temporarily) to force the BUSY signal high, because it should not become low until the $\overline{\text{STROBE}}$ pulse ends. The I4 pin, which detects the STROBE signal, is then re-programmed to detect the trailing edge (rising edge at the Centronics connector, but falling edge at pin I4 due to an inverting buffer). If the trailing edge already has occurred, then this reprogramming will set another interrupt pending immediately. There is, however, a possibility that the strobe edge could occur simultaneously with the reprogramming, with unknown results. For this reason, the STROBE signal is sampled by the firmware, and if the pulse has already completed, then instead of returning from the interrupt it jumps immediately to interrupt routine ''cstrbt'', which processes the trailing edge.

The code at label ''cstrbt'' is entered whenever either a trailing edge interrupt is detected on pin I4 (STROBE), or the leading edge interrupt routine jumps to it. It reprograms the I4 pin to detect a leading edge again, clears the I4 interrupt (which is automatically cleared only on interrupt service), then jumps to the ''setcen'' subroutine, which manipulates the BUSY and $\overline{\text{ACK}}$ signals appropriately, according to the contents of the ''cps'' variable and the selected $\overline{\text{ACK}}$ timing mode in variable ''ackmd''.

5.1.5.1 Centronics Error Handling

A buffer overrun error is processed at label ''cenerr''. This is the only kind of character error that can happen on a Centronics interface, and it would be due to an incorrect connection or a software error.

For internal firmware debugging purposes, the ''cps'' variable bit ''cbusy'' is again set to ensure that the Centronics interface will keep the BUSY signal set.

If an error is already waiting to be reported (bit ''aerr'' of variable ''alert'' is already set), then this is a ''multiple error'' condition, and cannot be fully reported. Instead, at label ''cenmer'', the bit ''errovf'' in variable ''errfgs'' is set. This variable is sent to the CPU when the error is reported. Also, the I4 interrupt is disabled, to prevent any further STROBE interrupts until a new SELECT-CENT command is received from the CPU.

If no error is waiting to be reported, then bit ''aerr'' of variable ''alert'' is set, requesting the main program to generate an !ERROR interrupt to the CPU. Further data is provided to be passed to the CPU:

variable ''errfgs'' is initialized to indicate only a buffer overrun error.

variable ''errchr'' is loaded with the character that was received and could not fit in the buffer.

Because the received character is reported with the error interrupt, and because no data is lost yet, the Centronics port is not disabled by this condition.

5.1.6 UART Communication

UART communication is performed by the UART interrupt routine at label ''uarint''. After pushing the required registers onto the stack, the routine determines which interface is selected. If it is the Centronics port, the only cause of the interrupt is the INPUT PRIME signal, and the HPC jumps to label ''uarprm'' (see Background Processing/Monitoring Tasks, below). If the UART port is selected, then it is due to either a receiver or a transmitter interrupt (and the INPUT PRIME is gated so that it cannot be presented).

5.1.6.1 UART Output

At label ''uarout'', a transmitter interrupt has been received.

If the bit ''icpu'' in variable ''ups'' is set, this means that the character just transmitted was a character sent by a CPU SEND-UART command, and the CPU is notified by requesting the !ACK-UART interrupt from the Main Program.

The subroutine ''setuar'' is now called, to determine whether any more characters need to be sent, either for XON/XOFF handshaking or because the CPU has requested the HPC to send another character. If so, another character is sent by ''setuar'', and the UART transmitter interrupt remains enabled. If not, the ''setuar'' routine disables the transmitter interrupt.

5.1.6.2 UART Input

At label ''uartin'', an interrupt has been generated by the UART receiver. This means that a character is available to be placed into the Communication Buffer.

The first action taken by the HPC is to read the receiver status register ENUR (which contains the 9th data bit and the Data Overrun and Framing Error error flags), then it reads the character itself from the RBUF register. The ENUR register is saved temporarily in variable ''enrimg'' for future processing, but is also held in the Accumulator, which is used here to ''accumulate'' error flags. The HPC then prepares to check for a parity error.

Parity checking is not a hardware feature of the HPC's UART, so a bit-table lookup is performed using the ''X,[B].b'' addressing mode of the IFBIT instruction. This addressing mode is similar to NS32000 bit addressing, in that it allows one to address up to 64 kbits (addressed from the contents of the X register) from a base address given in the B register. By placing the character to be checked into the X register, and pointing the B register at a properly constructed table (labels ''evntbl'' and ''oddtbl''), a parity error can be detected in a single IFBIT instruction (see for example label ''u8dopr'').

After loading the X and B registers, a multi-way branch is performed (jid), which branches to one of 8 labels depending on the character framing mode variable ''uframe'' (which is loaded by the SELECT-UART command). Each mode handles parity differently: labels ''uiod8'' and ''uiev8'' check for odd or even parity, respectively, including 9 character bits (8 data plus 1 parity) to make the test. Labels ''uiod7'' and ''uiev7'' include only 8 bits (7 data plus 1 parity). Label ''nopar'' handles the cases where no parity is included in the character frame. Also within these routines, a decision is made whether a Framing Error seen in the character is also a Break condition: if two consecutive characters are seen with framing errors with all zeroes in their parity and data fields, then the second character is reported as a Break character as well as having a framing error. If, at label ''uinpok'', no errors have been flagged in the Accumulator, the routine branches to label ''uingd'' to place the character into the Data Communication Buffer for the CPU. If errors have been discovered, then the character is instead reported to the CPU using the !DATA-ERR at label ''uinerc''.

The ''uingd'' portion of this routine is very similar to the portion of the Centronics input routine that places characters into the buffer for the CPU. A different mechanism is used for flow control, of course, to stop the external system if the buffer becomes full.

At label ''uinerc'', a check is made to determine whether the CPU has received the last character error reported. If not, this is a ''multiple error'' condition, handled at label ''uinmce''. If so, then this is reported as a new error at label ''uin1ce''. The error character and its error flags are provided to the Main Program in the mailboxes ''errchr'' and ''errfgs'', and the bit ''aerr'' in variable ''alert'' is set to request that a !DATA-ERR interrupt be sent to the CPU.

On a multiple-error condition, the new error flags are ORed with the old ones, handshaking is used to stop the external host system from sending more characters, and the UART receiver is automatically disabled. The CPU must issue a new SELECT-UART command to re-enable it.

Another pair of routines report an error if the buffer overflows. This error is reported at label ''uin1ef'' if no other error report is pending, or at label ''uinmef'' if this is a multiple error condition. On a multiple error, an attempt is made to stop the external host system from sending characters, and the UART receiver is disabled until the CPU issues a SELECT-UART command. (A single error does not disable the receiver, because no data has been lost yet: the !DATA-ERR interrupt reports the character with the error report.)

5.1.7 Buffer Status Reporting

For internal debugging purposes, four unassigned signals from the LCD Contrast Latch are updated to show the status of the buffer. While the buffer is full enough to pass to the CPU, one bit of the latch (IC 25G, pin 12) is high. While the buffer is full enough that the external system should stop, pin 15 is high. While the CPU is not ready to receive data from the CPU, pin 16 is high. If a buffer overrun condition occurs, and data is lost, or if any fatal error occurs (with a hexadecimal code appearing on the LCD display), then pin 19 goes high. The code that handles these bits is flagged with the word ''DEBUG'' in the comment field.

5.1.8 Background Processing/Monitoring Tasks

These are tasks that are not triggered directly by CPU commands.

Real-Time Clock (T1) Timer T1 is loaded with a constant interval value which is used to interrupt the HPC at 10 ms intervals. When the Timer T1 interrupt occurs (labels ''tmrint'', ''t1poll'', ''t1int''), and the real-time interrupt is enabled, the variable ''rtccnt'' is decremented to determine whether a !RTC interrupt should be issued to the CPU. If so, the bit ''artc'' in the ''alert'' word is set, requesting the main program to send a !RTC interrupt to the CPU. The main program, at label ''sndrtc'', interrupts the CPU. No other data is passed to the CPU.

At label ''kbdchk'' the panel pushbutton switches are also sampled. This process is described fully in Application Note AN-550.

At label ''dsrchk'', the state of the UART DSR flag is checked if the UART is selected and DSR monitoring mode has been requested by the CPU. If it has changed, this routine requests the Main Program to issue a !UART-STATUS

interrupt to the CPU. The UART receiver is also enabled and disabled by the state of this signal if DSR monitoring has been requested. (The CPU does not have to react to the interrupt for normal operation, but might wish to record its occurrence.)

At label ''brkchk'', if the UART is selected, and a BREAK has been detected, the UART data input pin is polled to determine whether the BREAK condition has ended. If a BREAK has ended, then this routine requests the Main Program to issue a !UART-STATUS interrupt to the CPU.

Centronics INPUT PRIME When the $\overline{\text{EXUI}}$ pin on the HPC is activated, and the Centronics port is selected rather than the UART, the UART service routine (at label ''uarprm'') sets bit ''aprime'' in the ''alert'' variable, requesting the main program to send a !PRIME interrupt to the CPU. The Centronics port is internally flagged (in the ''cps'' variable) as being ''busy'', and the Centronics Control Latch is updated to set the BUSY signal high. The UART interrupt is then disabled until a SELECT-CENT command is received from the CPU. In the main program, the !PRIME interrupt is sent to the CPU at label ''sndprm''. No other data is sent.

5.2 HPC Firmware Listing

```
        # Centronics Port input / checksum calculation / LCD output.
        #       Accepts up to 1024 characters on Centronics port,
        #       accumulates 8-bit checksum, and on receiving Ctrl-D,
        #       displays checksum on LCD display.

        .globl  start,main
        .globl  dataint,rtcint,primeint
        .globl  lcdint
        .globl  swint,usttsint,errint,uwrint
        .globl  diagint,badint

        .set    hpcctrl,0xFFFC00        # HPC Control/Status I/O location.
        .set    hpcdata,0xFFFE00        # HPC Data I/O location.
        .set    hpcpoll,0xFD0000        # HPC Poll address (UPIC).

        .set    cr,0xD
        .set    lf,0xA
        .set    ctrlD,'D'-0x40

start:
                                # Fill interrupt vector locations.

        addr    badint,vex              # Interrupt NMI.   (Unimplemented)
        addr    dataint,vex+4           # Interrupt 0x10.  Comm Buffer data.
        addr    rtcint,vex+8            # Interrupt 0x11.  Real-Time Clock.
        addr    badint,vex+12           # Interrupt 0x12.  (Unimplemented)
        addr    primeint,vex+16         # Interrupt 0x13.  Centronics PRIME.
        addr    badint,vex+20           # Interrupt 0x14.  (Unimplemented)
        addr    badint,vex+24           # Interrupt 0x15.  (Unimplemented)
        addr    badint,vex+28           # Interrupt 0x16.  (Unimplemented)
        addr    lcdint,vex+32           # Interrupt 0x17.  LCD data written.
        addr    swint,vex+36            # Interrupt 0x18.  Pushbutton event.
        addr    usttsint,vex+40         # Interrupt 0x19.  UART Status change.
        addr    errint,vex+44           # Interrupt 0x1A.  Error detected.
        addr    uwrint,vex+48           # Interrupt 0x1B.  UART Write ack.
        addr    badint,vex+52           # Interrupt 0x1C.  (Unimplemented)
        addr    diagint,vex+56          # Interrupt 0x1D.  Diagnostic.
        addr    badint,vex+60           # Interrupt 0x1E.  (Unimplemented)
        addr    badint,vex+64           # Interrupt 0x1F.  (Unimplemented)
        addr    badint,vex+68           # Interrupt 0x20.  (Unimplemented)
        addr    badint,vex+72           # Interrupt 0x21.  (Unimplemented)

        movb    $0,hpcctrl      # INITIALIZE command.
        movb    $100,hpcdata    # RTC value:  once per second.

        movb    $0x0B,hpcctrl   # Turn on two LED's to show we're alive.
        movb    $0x06,hpcdata

        movb    $1,hpcctrl      # Select Centronics port.
        movb    $1,hpcdata      #       BUSY drops during ACK/ pulse.
        movb    $100,hpcdata    #       Accept 100 characters before passing
                                #       buffer to CPU;
        movb    $120,hpcdata    #       Apply flow control if buffer has 120
                                #       characters.
```

TL/DD/9977–6

```
run:
        bispsrw $0x800          # Enable interrupts from HPC.

main:                   # Main program starts here.

        movd    datoptr,r1      # Register R1 contains buffer out pointer.

mwait: cmpd     datiptr,r1    # Wait here for a block to come in.
        bls     mwait

        movb    0(r1),r0        # Here, process character.
        cmpb    r0,$ctrlD       # if End of File, go type checksum.
        beq     typout

        addb    r0,ckdata
        addqd   $1,r1
        br      mwait

typout:                         # Send checksum out on LCDs.
        bicpsrw $0x800          # Disable interrupts.

        cbitb   $0,poutflg      # Clear LCD output acknowledge flag.

        movb    $0xA,hpcctrl    # Send-LCD command.
        movb    $0x6,hpcdata
        movb    $3,hpcdata

        movb    $0x1,hpcdata    # Clear panel LCD's.

        movzbd  ckdata,r0       # Send first hex character.
        lshd    $-4,r0
        movb    asctab[r0:b],r0
        movb    r0,hpcdata

        movzbd  ckdata,r0       # Send second hex character.
        andb    $0xF,r0
        movb    asctab[r0:b],r0
        movb    r0,hpcdata

        bispsrw $0x800          # Re-enable interrupts.

pnlout:         tbitb   $0,poutflg
        bfc     pnlout

        movqb   0,ckdata
        movd    $databuf,datiptr
        movd    datoptr,r1

        br      mwait   # Close loop:  infinite.

        ret     0       # End of main program.


maindat:        # Data for Main Program.

datiptr: .double databuf  # Pointer to Data Buffer area.
datoptr: .double databuf  # Pointer to Data Buffer area.
poutflg: .byte  1       # UART Output Ready.
ckdata:  .byte 0        # Accum. checksum.
asctab:          .byte  '0','1','2','3','4','5','6','7'
```

TL/DD/9977–7

```
        .byte  '8','9','a','b','c','d','e','f'
databuf: .blkb  1024   # Data buffer area.


        # Start of Interrupt Service Routines.
        # Invoked by ROM interrupt service.  Registers R0..R2 are already
        #   saved, but no ENTER instruction has been performed yet.

dataint:        # Interrupt 0x10.  Comm Buffer ready.

        movzbd  hpcdata,r0      # Get character count from HPC.
        movd    datiptr,r1

datalp:         movb    hpcdata,0(r1)   # Loop:  get character from HPC,
        addqd   1,r1            #        increment buffer address,
        acbd    -1,r0,datalp    #        decrement count and loop.

        movd    r1,datiptr
        ret     0

rtcint:                 # Interrupt 0x11.  Real-Time Clock.

        movb    $4,hpcctrl      # Send Flush-Buf command to HPC.
        ret     0

primeint:       # Interrupt 0x13.  Centronics PRIME.

        movb    $1,hpcctrl
        movb    $1,hpcdata
        movb    $100,hpcdata
        movb    $120,hpcdata
        ret     0

lcdint:                 # Interrupt 0x17.  LCD data written.

        sbitb   $0,poutflg
        ret     0

swint:          # Interrupt 0x18.  Pushbutton event.

        br      badint
        ret     0

usttsint:       # Interrupt 0x19.  UART Status change.

        br      badint
        ret     0

errint:                 # Interrupt 0x1A.  Error detected.

        br      badint
        ret     0

uwrint:                 # Interrupt 0x1B.  UART Write ack.

        br      badint
        ret     0

diagint:        # Interrupt 0x1D.  Diagnostic.
```

TL/DD/9977–8

```
        movb    hpcdata,r0
        movb    hpcdata,r0
        movb    hpcdata,r0
        movb    hpcdata,r0
        movb    hpcdata,r0
        ret     0

badint:                 # Trap for unimplemented interrupts.

        ret     0
```

TL/DD/9977–9

```
        # UART Port input / checksum calculation / UART output.
        #       Accepts up to 1024 characters on UART port,
        #       accumulates 8-bit checksum, and on receiving Ctrl-D,
        #       displays checksum by sending out on RS-232 port.

        .globl  start,main
        .globl  dataint,rtcint,primeint
        .globl  lcdint
        .globl  swint,usttsint,errint,uwrint
        .globl  diagint,badint

        .set    hpcctrl,0xFFFC00        # HPC Control/Status I/O location.
        .set    hpcdata,0xFFFE00        # HPC Data I/O location.
        .set    hpcpoll,0xFD0000        # HPC Poll address (UPIC).

        .set    cr,0xD
        .set    lf,0xA
        .set    ctrlD,'D'-0x40

start:
                                # Fill interrupt vector locations.

        addr    badint,vex              # Interrupt NMI.   (Unimplemented)
        addr    dataint,vex+4           # Interrupt 0x10.  Comm Buffer data.
        addr    rtcint,vex+8            # Interrupt 0x11.  Real-Time Clock.
        addr    badint,vex+12           # Interrupt 0x12.
        addr    primeint,vex+16         # Interrupt 0x13.  Centronics PRIME.
        addr    badint,vex+20           # Interrupt 0x14.
        addr    badint,vex+24           # Interrupt 0x15.
        addr    badint,vex+28           # Interrupt 0x16.
        addr    lcdint,vex+32           # Interrupt 0x17.  LCD data written.
        addr    swint,vex+36            # Interrupt 0x18.  Pushbutton event.
        addr    usttsint,vex+40         # Interrupt 0x19.  UART Status change.
        addr    errint,vex+44           # Interrupt 0x1A.  Error detected.
        addr    uwrint,vex+48           # Interrupt 0x1B.  UART Write ack.
        addr    badint,vex+52           # Interrupt 0x1C.  (Unimplemented)
        addr    diagint,vex+56          # Interrupt 0x1D.  Diagnostic.
        addr    badint,vex+60           # Interrupt 0x1E.  (Unimplemented)
        addr    badint,vex+64           # Interrupt 0x1F.  (Unimplemented)
        addr    badint,vex+68           # Interrupt 0x20.  (Unimplemented)
        addr    badint,vex+72           # Interrupt 0x21.  (Unimplemented)

        movb    $0,hpcctrl      # INITIALIZE command.
        movb    $100,hpcdata    # RTC value:  once per second.

        movb    $0x0B,hpcctrl   # Turn on two LED's to show we're alive.
        movb    $0x06,hpcdata

        movb    $2,hpcctrl      # Select UART and set up parameters.
        movb    $5,hpcdata      #       9600 baud,
        movb    $2,hpcdata      #       8 bits, no parity,
        movb    $0xA,hpcdata    #       XON/XOFF protocol, DTR always on.
        movb    $100,hpcdata    #       Accept 100 characters before passing
                                #       buffer to CPU;
        movb    $120,hpcdata    #       Apply flow control if buffer has 120
```

TL/DD/9977–10

```
                                #       characters.

run:
        bispsrw $0x800          # Enable interrupts from HPC.

main:                   # Main program starts here.

        movd    datoptr,r1      # Register R1 contains buffer out pointer.

mwait: cmpd     datiptr,r1    # Wait here for a block to come in.
        bls     mwait

        movb    0(r1),r0        # Here, process character.
        cmpb    r0,$ctrlD       # if End of File, go type checksum.
        beq     typout

        addb    r0,ckdata
        addqd   $1,r1
        br      mwait

typout:                         # Send checksum out on RS-232 port.
        movb    $cr,r0
        bsr     serout

        movb    $lf,r0
        bsr     serout

        movzbd  ckdata,r0
        lshd    $-4,r0
        movb    asctab[r0:b],r0
        bsr     serout

        movzbd  ckdata,r0
        andb    $0xF,r0
        movb    asctab[r0:b],r0
        bsr     serout

        movb    $cr,r0
        bsr     serout

        movb    $lf,r0
        bsr     serout

        movqb   0,ckdata
        movd    $databuf,datiptr
        movd    datoptr,r1

        br      mwait   # Close loop:  infinite.

        ret     0       # End of main program.

serout:         tbitb   $0,uoutflg
        bfc     serout

        cbitb   $0,uoutflg      # Indicate UART not ready.

        bicpsrw $0x800          # Critical Sequence:
        movb    $0xD,hpcctrl    #  Give Send-UART command to HPC.
        movb    r0,hpcdata      #  Give character to HPC.
        bispsrw $0x800          # End critical sequence.
```

TL/DD/9977–11

```
        ret     0


maindat:        # Data for Main Program.

datiptr: .double databuf  # Pointer to Data Buffer area.
datoptr: .double databuf  # Pointer to Data Buffer area.
uoutflg: .byte  1       # UART Output Ready.
ckdata:  .byte 0        # Accum. checksum.
asctab:          .byte  '0','1','2','3','4','5','6','7'
         .byte  '8','9','a','b','c','d','e','f'
databuf: .blkb  1024    # Data buffer area.



        # Start of Interrupt Service Routines.
        # Invoked by ROM interrupt service.  Registers R0..R2 are already
        #   saved, but no ENTER instruction has been performed yet.

dataint:        # Interrupt 0x10.  Comm Buffer ready.

        movzbd  hpcdata,r0      # Get character count from HPC.
        movd    datiptr,r1

datalp:         movb    hpcdata,0(r1)   # Loop:  get character from HPC,
        addqd   1,r1            #        increment buffer address,
        acbd    -1,r0,datalp    #        decrement count and loop.

        movd    r1,datiptr
        ret     0

rtcint:                 # Interrupt 0x11.  Real-Time Clock.

        movb    $4,hpcctrl      # Send Flush-Buf command to HPC.
        ret     0

primeint:       # Interrupt 0x13.  Centronics PRIME.

        br      badint
        ret     0

lcdint:                 # Interrupt 0x17.  LCD data written.

        br      badint
        ret     0

swint:          # Interrupt 0x18.  Pushbutton event.

        br      badint
        ret     0

usttsint:       # Interrupt 0x19.  UART Status change.

        br      badint
        ret     0

errint:                 # Interrupt 0x1A.  Error detected.

        br      badint
```

TL/DD/9977–12

```
        ret     0

uwrint:                 # Interrupt 0x1B.  UART Write ack.

        sbitb   $0,uoutflg
        ret     0

diagint:        # Interrupt 0x1D.  Diagnostic.
        movb    hpcdata,r0
        movb    hpcdata,r0
        movb    hpcdata,r0
        movb    hpcdata,r0
        movb    hpcdata,r0
        ret     0

badint:                 # Trap for unimplemented interrupts.

        ret     0
```

TL/DD/9977–13

```
        .title  CENTUART,'HPC FIRMWARE:  CENTRONICS/UART PORTS'
;
; program centuart.asm version 1.0     05/22/88
; Copyright (C) 1988 by National Semiconductor Corp.
;(**************************************************************************)
;(*                                                                        *)
;(*    Copyright (c) 1988       by National Semiconductor Corporation      *)
;(*                                                                        *)
;(*            National Semiconductor Corporation                                  *)
;(*    2900 Semiconductor Drive                                            *)
;(*    Santa Clara, California 95051                                       *)
;(*                                                                        *)
;(*    All rights reserved                                                 *)
;(*                                                                        *)
;(*    This software is furnished under a license and may be used          *)
;(*    and copied only in accordance with the terms of such license        *)
;(*    and with the inclusion of the above copyright notice. This          *)
;(*    software or any other copies thereof may not be provided or         *)
;(*    otherwise made available to any other person. No title to and       *)
;(*    ownership of the software is hereby transferred.                    *)
;(*                                                                        *)
;(*    The information in this software is subject to change without       *)
;(*    notice and should not be construed as a commitment by National      *)
;(*    Semiconductor Corporation.                                          *)
;(*                                                                        *)
;(*    National Semiconductor Corporation assumes no responsibility        *)
;(*    for the use or reliability of its software on equipment             *)
;(*    configurations which are not supported by National                  *)
;(*     Semiconductor Corporation.                                         *)
;(*                                                                        *)
;(**************************************************************************)
;
;       Derived from hpcupi.asm file.  However, commands have
;       been re-mapped (different code values), and so are not exactly
;       upward compatible.
;
;       Adds commands and interrupts to support input, buffering,
;       handshaking and mode selection for an RS-232 port and
;       a Centronics-style parallel port.
;

        .form   'Declarations:  Register Addresses'

psw     =       x'C0:w  ; PSW register
al      =       x'C8:b  ; Low byte of Accumulator.
ah      =       x'C9:b  ; High byte of Accumulator.
bl      =       x'CC:b  ; Low byte of Register B.
bh      =       x'CD:b  ; High byte of Register B.
xl      =       x'CE:b  ; Low byte of Register X.
xh      =       x'CF:b  ; High byte of Register X.

enir    =       x'D0:b
irpd    =       x'D2:b
ircd    =       x'D4:b
sio     =       x'D6:b
porti   =       x'D8:b
```

TL/DD/9977–14

```
obuf   =       x'E0:b  ; (Low byte of PORTA.)
portah =       x'E1:b  ; High byte of PORTA.
portb  =       x'E2:w
portbl =       x'E2:b  ; Low byte of PORTB.
portbh =       x'E3:b  ; High byte of PORTB.
upic   =       x'E6:b
ibuf   =       x'F0:b  ; (Low byte of DIRA.)
dirah  =       x'F1:b  ; High byte of DIRA.
dirb   =       x'F2:w
dirbl  =       x'F2:b  ; Low byte of DIRB.
dirbh  =       x'F3:b  ; High byte of DIRB.
bfun   =       x'F4:w
bfunl  =       x'F4:b  ; Low byte of BFUN.
bfunh  =       x'F5:b  ; High byte of BFUN.

portd  =       x'0104:b
enu    =       x'0120:b
enui   =       x'0122:b
rbuf   =       x'0124:b
tbuf   =       x'0126:b
enur   =       x'0128:b

t4     =       x'0140:w
r4     =       x'0142:w
t5     =       x'0144:w
r5     =       x'0146:w
t6     =       x'0148:w
r6     =       x'014A:w
t7     =       x'014C:w
r7     =       x'014E:w
pwmode =       x'0150:w
pwmdl  =       x'0150:b ; Low byte of PWMODE.
pwmdh  =       x'0151:b ; High byte of PWMODE.
portp  =       x'0152:w
portpl =       x'0152:b ; Low byte of PORTP.
portph =       x'0153:b ; High byte of PORTP.
eicon  =       x'015C:b

t1     =       x'0182:w
r1     =       x'0184:w
r2     =       x'0186:w
t2     =       x'0188:w
r3     =       x'018A:w
t3     =       x'018C:w
divby  =       x'018E:w
divbyl =       x'018E:b ; Low byte of DIVBY.
divbyh =       x'018F:b ; High byte of DIVBY.
tmmode =       x'0190:w
tmmdl  =       x'0190:b ; Low byte of TMMODE.
tmmdh  =       x'0191:b ; High byte of TMMODE.
t0con  =       x'0192:b


       .form   'Declarations:  Register Bit Positions

; Name     Position      Register(s)
; ----      --------      -----------

gie    =       0       ; enir
i2     =       2       ; enir, irpd, ircd
```

TL/DD/9977–15

```
i3      =       3       ; enir, irpd, ircd
i4      =       4       ; enir, irpd, ircd
tmrs    =       5       ; enir, irpd
uart    =       6       ; enir, irpd
ei      =       7       ; enir, irpd

dsr     =       7       ; porti only:  poll UART DSR.

uwmode  =       1       ; ircd
uwdone  =       0       ; irpd

tbmt    =       0       ; enu
rbfl    =       1       ; enu
b8or9   =       4       ; enu
xbit9   =       5       ; enu
wakeup  =       2       ; enur
rbit9   =       3       ; enur
frmerr  =       6       ; enur
doeerr  =       7       ; enur
eti     =       0       ; enui
eri     =       1       ; enui
xtclk   =       2       ; enui
xrclk   =       3       ; enui
b2stp   =       7       ; enui

wrrdy   =       0       ; upic
rdrdy   =       1       ; upic
la0     =       2       ; upic
upien   =       3       ; upic
b8or16  =       4       ; upic

t0tie   =       0       ; tmmdl
t0pnd   =       1       ; tmmdl
t0ack   =       3       ; tmmdl
t1tie   =       4       ; tmmdl
t1pnd   =       5       ; tmmdl
t1stp   =       6       ; tmmdl
t1ack   =       7       ; tmmdl
t2tie   =       0       ; tmmdh
t2pnd   =       1       ; tmmdh
t2stp   =       2       ; tmmdh
t2ack   =       3       ; tmmdh
t3tie   =       4       ; tmmdh
t3pnd   =       5       ; tmmdh
t3stp   =       6       ; tmmdh
t3ack   =       7       ; tmmdh

t4tie   =       0       ; pwmdl
t4pnd   =       1       ; pwmdl
t4stp   =       2       ; pwmdl
t4ack   =       3       ; pwmdl
t5tie   =       4       ; pwmdl
t5pnd   =       5       ; pwmdl
t5stp   =       6       ; pwmdl
t5ack   =       7       ; pwmdl
t6tie   =       0       ; pwmdh
t6pnd   =       1       ; pwmdh
t6stp   =       2       ; pwmdh
t6ack   =       3       ; pwmdh
t7tie   =       4       ; pwmdh
```

TL/DD/9977–16

```
t7pnd  =       5       ; pwmdh
t7stp  =       6       ; pwmdh
t7ack  =       7       ; pwmdh

t4out  =       0       ; portpl
t4tfn  =       3       ; portpl
t5out  =       4       ; portpl
t5tfn  =       7       ; portpl
t6out  =       0       ; portph
t6tfn  =       3       ; portph
t7out  =       4       ; portph
t7tfn  =       7       ; portph

cenclk =       0       ; portph (CCTLCLK signal).


txd    =       0       ; portbl, dirbl, bfunl
dtr    =       1       ; portbl, dirbl
pnlclk =       7       ; portbl, dirbl

lcvclk =       1       ; portbh, dirbh
; ua0 would be  2   , but requires no setup.
uwrrdy =       3       ; portbh, dirbh, bfunh
cdata  =       4       ; portbh (enables Centronics data to Port D).
astts  =       5       ; portbh (enables panel switches to Port D).
ledclk =       6       ; portbh, dirbh
urdrdy =       7       ; portbh, dirbh, bfunh

;      CONSTANTS
;
xon=   x'11    ; XON character:   Control-Q
xoff=  x'13    ; XOFF character:  Control-S


       .form   'Space Declarations'

botad= x'40    ; First address in buffer.
topad= x'BF    ; Last address in buffer.
bufsiz=        topad-botad+1   ; Length of buffer.
;
       .sect   BUFFER,BASE,ABS=botad   ; Data Communication Buffer.

       .dsb    bufsiz

       .endsect


       .sect   DSECT,BASE,REL  ; Basepage RAM variables (addresses 0000-00BF)

; WORD-ALIGNED
dummy: .dsw 1  ; x'00,01       ; Destroyed on reset (address 0).
       .set    upicsv,dummy    ; Temporary image of UPIC register.
alert: .dsw 1  ; Alert status bits to main program:
               ;  generate interrupts to CPU.
       .set    alerth,alert+1  ; Declare top byte of ALERT word.
cpuad: .dsw 1  ; Current address within CPU command buffer.
cpubuf:        .dsw 4  ; Buffer for accepting command parameters from CPU.
lcdsix:        .dsw 1  ; Pointer into LCD character string buffer.

;BYTE-ALIGNED
```

TL/DD/9977–17

```
numchr:         .dsb 1  ; Number of characters currently in data buffer.
cadin: .dsb 1   ; Current input byte address in data buffer
                ;  (first empty loc.).
cadout:         .dsb 1  ; Current output byte address in data buffer.
pascnt:         .dsb 1  ; Number of characters before data buffer full enough to
                ;  transmit to CPU.
stpcnt:         .dsb 1  ; Number of characters before host system is told to stop
                ;  transmitting.
numout:         .dsb 1  ; Number of data characters (total) being sent to CPU in
                ;  current transfer.
cntout: .dsb 1  ; Number of data characters remaining to be sent to CPU in
                ;  current transfer.
bstat: .dsb 1   ; Buffer Status byte.
cps:    .dsb 1  ; Centronics Port Status byte
                ;  (image of control signals).
ackmd: .dsb 1   ; Acknowledge Timing Mode:  Position of ACK/ pulse edges
                ;  on Centronics port relative to BUSY falling edge.
curcmd:         .dsb 1  ; Current command byte from CPU being processed.
numexp:         .dsb 1  ; Number of parameter bytes expected before command processing
                ;  begins.
lcvs:   .dsb 1  ; Image of LCD Voltage (Contrast) latch setting;  needed with
                ;  LCD RS (PAUX0) signal coming from this latch.
fshlim:         .dsb 1  ; Flush limit count:  used to limit number of characters passed
                ;  to CPU when an error report is pending.
errchr:         .dsb 1  ; Holds character on which an error was detected.
errfgs:         .dsb 1  ; Holds error flags associated with error character.
lcdfgs:         .dsb 1  ; Holds flag bits for characters sent to Panel LCD display.
lcdnum:         .dsb 1  ; Number of characters to be sent to LCD display.
lcdsfg:         .dsb 1  ; Flag bits associated with characters in LCD String Buffer.
lcdsct:         .dsb 1  ; Counter for characters being sent to LCD display from String
                ;   Buffer.
swlast:         .dsb 1  ; Last-sampled switch values.
swlsnt:         .dsb 1  ; Last switch values sent to CPU.
beepct:         .dsb 1  ; Beep duration count.  Counts occurrences of T0 interrupt.
uframe:         .dsb 1  ; Frame mode for UART.
uflow: .dsb 1   ; Flow control mode for UART.
ups:    .dsb 1  ; UART Status byte.
uschr: .dsb 1   ; UART Send Character:  from CPU.
uinchr:         .dsb 1  ; UART Input Character:  character last received from UART.
enrimg:         .dsb 1  ; UART ENUR register image in memory.
rtcivl:         .dsb 1  ; Real-Time Clock Interval (units of 10 milliseconds).
rtccnt:         .dsb 1  ; Real-Time Clock Current Count (units of 10 milliseconds).
rtevs: .dsb 1   ; Events to check for on Timer T1 interrupts.
ustat: .dsb 1   ; UART status for CPU.
dsevc: .dsb 1   ; Diagnostic Interrupt:  Severity Code.
derrc: .dsb 1   ; Diagnostic Interrupt:  Error Code.
dbyte: .dsb 1   ; Diagnostic Interrupt:  Error Byte.
dccmd: .dsb 1   ; Diagnostic Interrupt:  Current Command.
dqual: .dsb 1   ; Diagnostic Interrupt:  Qualifier (Command Status).

; * Addresses 0040-00BF are reserved for the Data Communication Buffer
;    (128 bytes).

;       BIT POSITIONS

        ; Bits in BSTAT byte (Data Communication Buffer Status):
pass=   0       ; Data is ready to be passed to the CPU.
passng=         1       ; Indicates that some of the data in the buffer is being
                ;   passed to the CPU.
stop=   2       ; Indicates that host has been requested to stop transmitting.
```

TL/DD/9977–18

```
cpubsy=        3       ; Indicates that CPU is not able to receive any more data.
ifcbsy=        4       ; Indicates that the interface is considered busy by CPU.
full=  5       ; Indicates that the interface is completely full.  Any more
               ;   characters will overflow it.

       ; Bits in CPS (Centronics Port Status byte)
cack=  0       ; ACK/ Strobe.
cauxl= 1       ; AUXOUTl Signal.
cbusy= 2       ; BUSY Signal.
cselct=        3       ; SELECT Signal.
ccall= 4       ; CALL Signal.
cfault=        5       ; FAULT/ Signal.
caux2= 6       ; AUXOUT2 Signal.
enprm= 7       ; 1 enables INPUT PRIME/ interrupt from Centronics port.

       ; Bits in ACKMD (Centronics Acknowledge Mode byte)
; (Bits 0 and 1 give timing relationship between BUSY and ACK/.)
clinmd=        2       ; 1 = Centronics Line Mode.  Buffer limits must also both be 1.
; (Other bits unassigned.)

       ; ALERT status word (low-order byte) bits:
aflush =       0       ; Flush Data Buffer.
artc   =       1       ; Real-Time Interrupt detected.
aprime =       3       ; INPUT PRIME detected from Centronics interface.
alcdak =       7       ; LCD Panel Write Acknowledge.
       ; ALERT status word (high-order byte, named alerth) bits:
abutton        =       0       ; Pushbutton switch state change.
austat =       1       ; UART status change.
aerr   =       2       ; Error detected (UART or buffer overflow).
auack  =       3       ; UART output acknowledge:  character sent.
adiag  =       5       ; Diagnostic interrupt.
; (Other bits not defined.)

       ; ERRFGS error flags byte sent to CPU with !BAD-DATA interrupt:
doe=   7       ; Data Overrun Error on UART.
frm=   6       ; Framing Error on UART.
par=   5       ; Parity error on UART.
bufovf=        4       ; Buffer Overflow condition (flow control did not work).
errovf=        3       ; Error Overflow condition.  Two or more errors occurred
               ;   so close together that the first error could not be
               ;   reported before the second error occurred.  Details
               ;   of the second error are lost.
brk=   2       ; Break condition detected in addition to Framing error.
; (Other bits not defined.)

       ; CURCMD byte:  Current CPU command.  The lower 5 bits contain a code
       ;                 in the range 0-10 (hex).  The upper two bits contain
       ;                 further information about command collection:
cmdemp=        7       ; Bit 7 (MSB) of curcmd = 1 means that no command is being
               ;  processed and curcmd byte is "empty".
getcnt=        6       ; Bit 6 of curcmd = 1 means that the count is being received
               ;  for a variable-length command.

       ; LCVS byte:  LCD Voltage (Contrast) Latch memory image.
       ;               Contains voltage value in its least-significant 3 bits,
       ;               RS signal to LCD controller in bit 3, and debugging
       ;               information in its top 4 bits.
pnlrs= 3       ; Bit 3 is (inverted) RS signal to panel.

       ; UPS byte:  Status of UART output and flow control.
```

TL/DD/9977–19

```
usel=  7       ; When set, means that UART port is selected.
mcemd= 6       ; Receiver disabled due to multiple character error.
brkmd= 5       ; BREAK signal has been detected and is still active; receiver
               ;  disabled.
onebrk=        4       ; One character which is possibly a BREAK has been seen.
icpu=  3       ; When set, means that CPU should be informed of next UART
               ;  transmitter interrupt.
schr=  2       ; Request to send a character from uschr location (from CPU).
cus=   1       ; Current UART status:  1 = stopped.
luss=  0       ; Last UART Status Sent:  Indicates what the external system
               ;  thinks the UART's status is.

       ; UFLOW byte:  Modes for UART flow control.
flemp= 7       ; 1 = No flow control yet provided since reset.
;(intervening bits not defined.)
xonb=  3       ; 1 = XON/XOFF protocol mode selected.
dtrb1= 2       ; DTR Mode field:  00 = permanently low.
dtrb0= 1       ;                  01 = permanently high.
               ;                  10 = low when ready.
               ;                  11 = high when ready.
dsrb=  0       ; 1 = characters received while DSR low will not be accepted.

       ; USTAT byte:  Status of UART reported to CPU.
dsrflg=        0       ; State of DSR signal.  1 = Data Set Ready condition.
brkflg=        1       ; 1 = End of BREAK condition detected.

       ; RTEVS byte:  Events to check for at 10-millisecond intervals.
       ;              (T1 Underflows)
rtcenb=        0       ; 1 = Real-Time Clock interrupts enabled to CPU.
brkenb=        1       ; 1 = UART Break mode;  report end of break.


       .sect   STACK,RAM16,REL        ; On-chip RAM in addresses 01C0-01FF.
stackb:        .dsw    16      ; Space for 8 words beyond
                       ;  interrupt context.
avail: .dsw    12      ; Spare portion of this space.
lcdbuf:        .dsw    4       ; LCD String Buffer.

       .form   'Code Section'
       .sect   CSECT,ROM16,REL ; Code space.  (On-chip ROM)

       ; Declarations of subroutines called by one-byte JSRP instruction.

       .spt    rdwait          ; Waits for CPU to read a value from UPI port.
       .spt    wrpnl           ; Writes to LCD panel (for initialization only).

       ; Program starts at label "start" on reset.  This routine is the fatal
       ;  error handler, located here for convenience in setting breakpoint.

hangup:        rbit    gie,enir        ; Fatal error:  signal it and halt.
       sbit    7,lcvs          ; Signal error on most-significant bit of
                               ;  LCD Contrast Latch.
       sbit    pnlrs,lcvs      ; Select command mode for LCD controller.
       ld      portah,lcvs     ; Place error on Port A for latch.
       sbit    lcvclk,portbh   ; Clock LCD Contrast Latch high,
       rbit    lcvclk,portbh   ;  then low to load it.
       sbit    t6stp,pwmdh     ;
       rbit    t6tie,pwmdh     ; Set up Timer T6 for non-interrupt use.
       nop
       rbit    t6pnd,pwmdh     ; Clear Pending bit.
```

TL/DD/9977–20

```
        pop     0.w             ; Get error address from stack.
        ld      sp.w,#stackb    ; In case of stack underflow, re-initialize SP.
        ld      A,#x'01
        jsrl    wrpnl           ; Clear LCD panel.
        rbit    pnlrs,lcvs      ; Set up panel for data.
        ld      portah,lcvs     ; Place error on Port A for latch.
        sbit    lcvclk,portbh   ; Clock LCD Contrast Latch high,
        rbit    lcvclk,portbh   ;  then low to load it.
        ld      A,l.b           ; Process first character of return address.
        swap    A
        and     A,#x'0F
        ld      A,hextab[A].b
        jsrl    wrpnl           ; Display it on LCD panel.
        ld      A,l.b           ; Process second character of return address.
        and     A,#x'0F
        ld      A,hextab[A].b
        jsrl    wrpnl           ; Display it on LCD panel.
        ld      A,0.b           ; Process third character of return address.
        swap    A
        and     A,#x'0F
        ld      A,hextab[A].b
        jsrl    wrpnl           ; Display it on LCD panel.
        ld      A,0.b           ; Process last character of return address.
        and     A,#x'0F
        ld      A,hextab[A].b
        jsrl    wrpnl           ; Display it on LCD panel.

hgupi:  ifbit   rdrdy,upic      ; Check to see if OBUF register is full.
        ld      obuf,#vdiag     ; If not, fill it with !DIAG vector
                                ;  continuously.
        ifbit   i3,irpd         ; Check for UPI data ready.
        jp      hgupil
        jp      hgupi

hgupil:         ifeq    ibuf,#x'A5      ; Check for RESET command.
        jp      hgrst
        jp      hgupi2
hgrst:  ifbit   la0,upic
        jp      hgupi2
        jmpl    xreset          ; If so, then go reset the HPC.

                                ; This is part of the outer loop, waiting for
                                ;  the RESET command.
hgupi2:         ld      irpd,#x'F7      ; Clear the UWR detector,
        jp      hgupi           ;  and keep looking.  This is an
                                ;  infinite loop until RESET is seen.

hextab:         .byte   '0','1','2','3','4','5','6','7'
        .byte   '8','9','A','B','C','D','E','F'

        .form   'Hardware Initialization'

start:  ld      psw.b,#x'08     ; Set one WAIT state.

srfsh:                  ; Start dynamic RAM refreshing,
                        ;  as quickly as possible.
        sbit    t4out,portpl    ; Trigger first refresh
                                ;  immediately.
        sbit    t4stp,pwmdl     ; Stop timer T4 to
                                ;  allow loading,
```

TL/DD/9977–21

```
        ld      t4,#8           ;  then load it.
        rbit    t4stp,pwmdl     ; Start timer T4.
        sbit    t4tfn,portpl    ; Enable pulses out.
        ld      r4,#8           ; Load R4.

supi:                           ; Set up UPI port.
        ld      upic,#x'18      ; 8-Bit UPI Mode
                                ;  enabled.

        sbit    uwrrdy,bfunh    ; Enable UWRRDY/ out.
        sbit    uwrrdy,dirbh
        ld      A,ibuf          ; Empty IBUF register,
                                ;  in case of false trigger.

        sbit    urdrdy,bfunh    ; Enable URDRDY/ out.
        sbit    urdrdy,dirbh

                                ; Set up UREAD/ interrupt.
        sbit    i2,ircd         ; Detects rising edges.
        ld      irpd,#x'FB      ; Clear any false interrupt
                                ;  due to mode change.

                                ; Set up UWRITE/ interrupt.
        sbit    i3,ircd         ; Detects rising edges.
        ld      irpd,#x'F7      ; Clear any false interrupt
                                ;  due to mode change.

sram:                   ; Clear all RAM locations.
                        ; Clear Basepage bank:
        ld      BK,#x'0000,#x'00BE      ; Establish loop base and limit.
sraml1:         clr     A
        xs      A,[B+].w
        jp      sraml1

                        ; Clear Non-Basepage bank:
        ld      BK,#x'01C0,#x'01FE      ; Establish loop base and limit.
sraml2:         clr     A
        xs      A,[B+].w
        jp      sraml2

sskint:                         ; Set up Stack and remove
                        ;  individual interrupt enables.
        ld      sp.w,#stackb+2  ; Move stack to high
                                ;  bank of on-chip RAM.
        ld      stackb.w,#hangup  ; Safeguard against
                                  ;  stack underflow.
        ld      enir,#x'00      ; Disable interrupts
                                ;  individually.

tminit:         ld      t0con,#x'08
        ld      tmmode,#x'4440  ; Stop timers T1, T2, T3.
        ld      divby,#x'0055   ; UART set to 9600 Baud.
        ld      tmmode,#x'CCC8  ; Clear and disable timer
                                ;  T0-T3 interrupts.

scent:                          ; Set up Centronics parallel
                                ;  port.
        ld      dirah,#x'FF     ; Enable multiplexed outputs.
        sbit    astts,portbh    ; Enable and remove ENASTTS/ signal.
        sbit    astts,dirbh
```

TL/DD/9977–22

```
        sbit    cdata,portbh    ; Enable and remove ENCDATA/ signal.
        sbit    cdata,dirbh
        ld      cps,#x'25       ; Set up Port A data for
                                ;  Centronics Control.
        jsrl    setcen          ; Send to Centronics latch and to Busy flag.

                                ; Set up I4 interrupt on
        sbit    i4,ircd         ;  CINTR/ (rising edge).
        ld      irpd,#x'EF      ; Clear any false interrupt
                                ;  caused by mode change.

suart:                          ; Set up RS-232 port.
        sbit    txd,bfunl       ; Enable TXD output.
        sbit    txd,dirbl
        rbit    dtr,portbl      ; Set up DTR signal.  (State is arbitrary:
                                ;  low typically means not ready.)
        sbit    dtr,dirbl       ; Enable it as an output pin.
        ld      enu,#x'0        ; 8-bit Mode.
        ld      enur,#x'0       ; Clear Wake-Up Mode.
        ld      enui,#x'80      ; Internal baud;  2 stop
                                ;  bits;  no interrupts.

sled:   ld      portah,#x'FF    ; Set up to turn off LED's.
        rbit    ledclk,portbh   ; Start with LEDCLK low,
        sbit    ledclk,dirbh    ;  (enable output),
        sbit    ledclk,portbh   ;  then high,
        rbit    ledclk,portbh   ;  then low again.

stmrs:                          ; Set up remaining timers.
                                ;  (T1-T3 already stopped
                                ;   and pending bits cleared
                                ;   at tminit above.)

        ld      t1,#12287       ; T1 runs at 10-millisecond real-time interval.
        ld      r1,#12287
                                ; Timer remains stopped, and interrupt
                                ;  disabled, until INITIALIZE command.

        ld      pwmode,#x'4440  ; Stop timers T5-T7.
        nop                     ; Wait for valid PND
        nop                     ;  bits.
        ld      pwmode,#x'CCC8  ; Clear and disable
                                ;  interrupts from all
                                ;  PWM timers.

        ld      r6,#x'FFFF      ; No modulus for LCD Display Ready timer.

        ld      t7,#204 ; Set T7 to underflow at 6 KHz rate
        ld      r7,#204 ;  (= 3 KHz at pin).
        rbit    t7tfn,portph    ; Disable beep tone to panel speaker.
        rbit    t7stp,pwmdh     ; Start T7 running.


slcd:                   ; Set up LCD display.
                        ; Requires use of timer T6, so
                        ;  appears after timer initialization.

                        ; First, set up LCD contrast.
        ld      lcvs,#x'0A      ; Initialize memory image of LCD Voltage
                                ;  latch, containing RS (PAUX0) bit also.
```

TL/DD/9977–23

```
        ld      portah,lcvs     ; Arbitrary initial contrast level of 5,
                                ;  and RS/ (PAUX0/) is high (="command").
        rbit    lcvclk,portbh   ; Start with LCVCLK low,
        sbit    lcvclk,dirbh    ;  (enable output)
        sbit    lcvclk,portbh   ;  then high,
        rbit    lcvclk,portbh   ;  then low to get it into LCV latch.

                        ; Initialize PNLCLK (Panel "E" signal).
        sbit    pnlclk,portbl   ; Start with PNLCLK high
        sbit    pnlclk,dirbl    ;  (enable output).

                        ; Wait for worst-case command
                        ;  execution time (4.9 ms, twice), in case
                        ;  a panel command was triggered while
                        ;  PNLCLK was floating.
        sbit    t6ack,pwmdh     ; Clear T6 PND bit.
        ld      t6,#13000       ; Set T6 to twice 4.9 milliseconds.
        rbit    t6stp,pwmdh     ; Start timer T6.
lcdlpl: ifbit   t6pnd,pwmdh     ; Wait for T6 PND bit
                                ;  to be set.
        jp      lcdgol
        jp      lcdlpl
lcdgol: sbit    t6stp,pwmdh     ; Stop timer T6.
        sbit    t6ack,pwmdh     ; Clear T6 PND bit.

                        ; Reset Panel controller (per Hitachi HD44780
                        ;   User's Manual).

                        ; (Panel RS signal was set
                        ;  in LCD Contrast initialization above,
                        ;  so no change needed here to
                        ;  flag these as a commands.)

        ld      A,#x'38         ; Send "8-Bit Mode, 2 Lines" command:  one;
        jsrl    wrpnl
        ld      A,#x'38         ; two;
        jsrl    wrpnl
        ld      A,#x'38         ; three;
        jsrl    wrpnl
        ld      A,#x'38         ; four times.
        jsrl    wrpnl
        ld      A,#x'08         ; Disable display.
        jsrl    wrpnl
        ld      A,#x'01         ; Clear display RAM.
        jsrl    wrpnl

                        ; Initial default mode settings.

        ld      A,#x'06         ; Set mode to move cursor to the right, no
        jsrl    wrpnl           ;   automatic shifting of display.
        ld      A,#x'0E         ; Enable display:  non-blinking cursor mode.
        jsrl    wrpnl


;       CONTINUES TO MAIN PROGRAM INITIALIZATION
        .form   'Main Program Initialization'


minit:
                        ; Once-only initializations.
```

TL/DD/9977–24

```
        ld      curcmd,#x'80    ; Current Command:  top bit set means "none".
        ld      cpuad,#cpubuf   ; Set CPU command index to beginning of buffer.
        ld      numexp,#8       ; Arbitrary starting value.

                        ; Arbitrary set of initialization values for variables,
                        ;  in effect until receipt of the first INITIALIZE
                        ;  command.

        ld      numchr,#0       ; Clear count of characters received.
        ld      cadin,#botad    ; Next character in from comm port goes to
                                ;  first byte of buffer.
        ld      cadout,#botad   ; Next port data character out (to CPU)
                                ;  comes from first byte of buffer.
        ld      numout,#0       ; No characters being sent to CPU.
        ld      cntout,#0       ; No characters being sent to CPU.
        ld      pascnt,#125     ; Send to CPU when 125 characters received.
        ld      stpcnt,#126     ; Stop host when 126 characters received.
        ld      bstat,#0        ; Set buffer ready to receive.
        ld      alert,#0        ; No events pending.
        ld      ackmd,#1        ; BUSY will fall during ACK/ pulse.
        ld      errchr,#55      ; Arbitrary fill for error character.
        ld      errfgs,#0       ; Clear error detail flags.
        ld      uflow,#x'80     ; Set UART flow control mode byte empty.

runsys:                         ; Enable interrupts, start timers and go to main loop.

        sbit    tmrs,enir       ; Enable timer interrupts.  (Done here
                                ;  to allow certain commands without an
                                ;  INITIALIZE command first.)
        sbit    i3,enir         ; Enable CPU Command interrupt.
        sbit    gie,enir        ; Enable interrupt system.


        .form   'Main Scan Loop'

;       Declarations

vdata   =       x'10    ; CPU DATA vector number.
vrtc    =       x'11    ; Real-Time Clock vector number.
vprime  =       x'13    ; Centronics INPUT PRIME signal.
vlcdak  =       x'17    ; Acknowledge finished writing to LCD panel.
vbutton         =       x'18    ; Pushbutton status change: a button pressed or
                        ;  released.
vustat  =       x'19    ; Change in UART DSR signal, or end of BREAK.
verr    =       x'1A    ; Character received with error from UART, or gross
                        ;   error condition in buffering or flow control on
                        ;   either port.
vuack   =       x'1B    ; UART output acknowledge:  character sent.
vdiag   =       x'1D    ; Diagnostic Interrupt.

mainlp:
                ; Error Vectors for unimplemented or
                ;   unexpected interrupts.

        .ipt    1,hangup        ; NMI:   never expected.
        .ipt    2,hangup        ; UPI READ READY:  never expected.
        .ipt    7,hangup        ; EI:    never expected.

chkdta:
```

TL/DD/9977–25

```
        ld      A,bstat         ; Test state of buffer.
        and     A,#x'09         ; Check PASS and CPUBUSY bits.
        ifeq    A,#x'01         ; If PASS and not CPUBUSY,
        jsrl    snddta          ;  then go send a block of data to CPU.

chkalt:         ifeq    alert.w,#x'00   ; Check for alert conditions.
        jmpl    chkrsp          ; If none, go check for response ready.

        ifbit   artc,alert.b    ; Check for RTC interrupt request.
        jsrl    sndrtc          ; If so, then send Real-Time Clock Interrrupt.

        ifbit   aprime,alert.b  ; Check for Centronics Input Prime signal.
        jsrl    sndprm          ; If so, send Input Prime interrupt.

        ifbit   alcdak,alert.b  ; Check for LCD Panel write done.
        jsrl    sndlak          ; If so, then send LCD Acknowledge interrupt.

        ifbit   aflush,alert.b  ; Check for Flush Buffer request.
        jsrl    sndfsh          ; If so, then send data in buffer to CPU.

        ifbit   abutton,alerth.b ; Check for a pushbutton change.
        jsrl    sndbtn          ; If so, then report the change to the CPU.

        ifbit   austat,alerth.b ; Check for a UART status change.
        jsrl    sndust          ; If so, then report the change to the CPU.

        ifbit   aerr,alerth.b   ; Check for a data error condition.
        jp      cherr
        jp      nocher
cherr:  ifbit   cpubsy,bstat    ; Suppress if CPU busy.  (CPU needs to
        jp      nocher          ;  receive flushed characters first.)
        ifgt    fshlim,#0
        jsrl    sndfsh          ; If a flush is still needed, then do it first.
        jsrl    snderr          ; If so, then report the error to the CPU.
nocher:                                 ;  (This line deliberately empty.)

        ifbit   auack,alerth.b  ; Check for UART output done.
        jsrl    snduak          ; If so, then send UART-ACKNOWLEDGE interrupt.

        ifbit   adiag,alerth.b  ; Check for Diagnostic Interrupt.
        jsrl    sndiag          ; If so, then send interrupt and data.

chkrsp:         jmpl    chkdta          ; No "responses" defined yet;  just close loop.

        .form   'Main:  Send Real-Time Clock Interrupt'

; No data transfer;  just trigger interrupt and continue.

sndrtc:
        rbit    artc,alert.b    ; Clear ALERT bit.
        jsrl    rdwait          ; Check that UPI interface is ready.
                                ; If not, loop until it is.

        ld      obuf,#vrtc      ; Load Real-Time Clock vector into OBUF for CPU.
        ret                     ; Return to main loop.
```

TL/DD/9977–26

```
; No data transfer;  just trigger interrupt and continue.

sndlak:
       rbit    alcdak,alert.b  ; Clear ALERT bit.
       jsrl    rdwait          ; Check that UPI interface is ready.
                               ; If not, loop until it is.

       ld      obuf,#vlcdak    ; Load LCD-Acknowledge vector into OBUF for CPU.
       ret                     ; Return to main loop.

       .form   'Main:  Send Pushbutton Status to CPU'

sndbtn:
       jsrl    rdwait          ; Check that UPI interface is ready.
                               ; If not, loop until it is.

       ld      obuf,#vbutton   ; Load BUTTON-DATA vector into OBUF for CPU.

       jsrl    rdwait          ; Check that UPI interface is ready.
                               ; If not, loop until it is.

       rbit    gie,enir        ; *** Begin Indivisible Sequence ***
       ld      obuf,swlsnt     ; Load Pushbutton Data Byte into OBUF for CPU.
       rbit    abutton,alerth.b ; Clear ALERT bit.
       sbit    gie,enir        ; *** End Indivisible Sequence ***
       ret                     ; Return to main loop.


       .form   'Main:  Send Data from Data Buffer to CPU'

; Trashes A, B, K (limit), and C flag.  May trash X in future.

       ;  Buffer Flush request serviced here.
sndfsh:
       rbit    aflush,alert.b  ; Reset Flush request.
       ifeq    numchr,#0       ; If no characters to send, just return,
       ret                     ; else go to Send Data routine.
       jmpl    snddta

       ;  Automatic Pass condition serviced here.
snddta:
       ifbit   aerr,alerth.b   ; Check for a communication or buffer error.
       jp      chkflm          ; If so, there is a limit on the number of
                               ;  characters to send.  Investigate further.
       jp      snddl           ; Else, go ahead and perform automatic pass.

chkflm:        ifeq    fshlim,#0       ; Here, a flush limit is in effect due to an
       ret                     ;  error condition.  Check that the limit is
                               ;  non-zero before initiating the pass.  If
                               ;  zero, then simply return without passing.

snddl: jsrl    rdwait          ; Check that UPI interface is ready.
                               ; If not, loop until it is.

       ld      obuf,#vdata     ; Load DATA vector into OBUF for CPU.

       jsrl    rdwait          ; Check that UPI interface is ready
                               ;  (CPU has acknowledged DATA interrupt).
                               ; If not, loop until it is.
```

TL/DD/9977–27

```
        rbit    gie,enir        ; Indivisible operation:  disable interrupts
                                ;  momentarily.
        sbit    passng,bstat    ; Indicate data being passed to CPU.
        ld      numout,numchr   ; Sample number of characters in buffer.
                                  ;  This becomes the number of characters to
                                  ;  transfer,
        ifbit   aerr,alerth.b     ;  unless there is a flush limit in effect,
        ld      numout,fshlim     ;  in which case that limit is used.
        ld      fshlim,#0       ; Any flush limit is set to zero at this point,
                                ;  disabling any data passing until the error
                                ;  condition is reported.
                                ;  (This does not need to be conditional.)
        sbit    gie,enir        ; End indivisible operation:  re-enable
                                ;  interrupts.
        ld      obuf,numout     ; Give number of characters to CPU.
        ld      cntout,numout ; Copy number of characters to temporary
                                  ;  count location.

        ld      B,cadout        ; Initialize for loop below.
        ld      K,#topad        ; Establish buffer limit.

snddlp:                         ; Loop to send characters from data buffer to CPU.

        lds     A,[B+].b        ; Load from next byte in buffer, and increment
                                ;  address pointer in B.
        jp      sndd4           ; If skip occurs (incremented past end
        ld      B,#botad        ;  of buffer), reset pointer to top of buffer.

sndd4:  jsrl    rdwait          ; Check that UPI interface is ready.
                                ; If not, loop until it is.

        st      A,obuf          ; Give character to CPU.
        decsz   cntout          ; Check if last character.
        jp      snddlp          ;  No:  Loop.

                                ;  Yes: Update pointers and buffer status.
        ld      cadout,B.b      ; Update current pointer address in memory.

        rbit    gie,enir        ; *** Begin Indivisible Sequence. ***
        and     bstat,#x'FC     ; Clear PASS and PASSING flags.

        rbit    pass+4,lcvs     ; (DEBUG:  Update PASS in LCD Contrast latch.)
        ld      portah,lcvs
        sbit    lcvclk,portbh
        rbit    lcvclk,portbh

        sc                      ; (Set carry for subtraction.)
        subc    numchr,numout   ; Adjust number of characters in buffer to
                                ;  reflect those just removed.
        ld      A,#bufsiz       ; Check whether the buffer is any longer
        ifgt    A,numchr        ;   completely full.
        rbit    full,bstat      ; No:  remove FULL indication (if set).
        ifgt    A,numchr        ; (DEBUG:  update FULL for LCV latch.)
        rbit    7,lcvs
        ifbit   stop,bstat      ; Check whether host was stopped.
        jp      sdstp           ;  Yes:  continue,
        jmpl    sdend           ;  No:  terminate indivisible sequence and
                                ;       return to main loop.

sdstp:  ifgt    stpcnt,numchr   ; Check whether number of characters is
```

TL/DD/9977–28

```
                                ;  now less than "Stop" value to host.
        jp      sdstpl
        jmpl    sdend           ; If not, then return to main loop.

sdstpl:         rbit    stop,bstat      ; Clear "Stop Host" flag.
        rbit    5,lcvs

                                ; Check which port to enable for more data.
        ifbit   usel,ups        ; Check if UART is selected.
        jmpl    sdusts          ; If so, go set up flow control.
        ifbit   enprm,cps       ; Check if Centronics port is selected.
        jmpl    sdcsts          ; If so, go set up Centronics BUSY.
        jmpl    sdend           ; Otherwise, do nothing more and return.

sdcsts:         ifbit   clinmd,ackmd    ; Check if in Centronics Line Mode.  If so,
        jmpl    sdend           ;   the CPU itself must command the ACK action.
        ld      A,bstat         ; Test whether data communication with
                                ;   host should be allowed to continue.
        and     A,#x'3C         ;   Bits involved are STOP, CPUBSY, IFCBSY and
                                ;   FULL.
        ifeq    A,#x'00         ;   If no stop conditions are in effect,
        rbit    cbusy,cps       ;   clear the BUSY indication in CPS
                                ;   (Centronics Port Status) byte in memory.
        ifbit   i4,ircd         ; If not between the two interrupt services
                                ;   of a Centronics strobe, then
        jsrl    setcen          ;   call Centronics port control setup routine,
                                ;   to generate ACK/ pulse and clear BUSY.
                                ; (If this sequence does occur between the
                                ;  leading and trailing edge interrupts for
                                ;  STROBE/, then the trailing edge routine
                                ;  will pulse ACK/ when it is allowed to run.)
        jmpl    sdend

sdusts:         rbit    cus,ups         ; Set UART not busy.
        jsrl    dtron           ; Set DTR handshake appropriately.
        ifbit   eti,enui        ; Check if a UART transmitter interrupt will
                                ;  be occurring.
        jmpl    sdend           ; If so, then no further action is required.
        ifbit   xonb,uflow      ; Otherwise, if XON protocol is in effect,
        jsrl    setuar          ;  then check and perform flow control.
        jmpl    sdend           ; Then exit to main program.

sdend:
        ld      portah,lcvs     ; (DEBUG: Update LCV latch.)
        sbit    lcvclk,portbh
        rbit    lcvclk,portbh
        sbit    gie,enir        ; *** End Indivisible Sequence. ***

        ret                     ; Return to main program loop.

        .form   'Main:  Send Input Prime interrupt to CPU'

sndprm:                         ; Send INPUT PRIME interrupt to CPU.
        rbit    aprime,alert.b  ; Clear ALERT bit.
        jsrl    rdwait          ; Check that UPI interface is ready.
                                ; If not, loop until it is.

        ld      obuf,#vprime    ; Load PRIME vector into OBUF for CPU.
        ret                     ; Return to main program loop.
```

TL/DD/9977–29

```
        .form   'Main:  Report a UART DSR change or END OF BREAK'

sndust:
        jsrl    rdwait          ; Check that UPI interface is ready.
                                ; If not, loop until it is.

        ld      obuf,#vustat    ; Load UART-STATUS vector into OBUF for CPU.

        jsrl    rdwait          ; Check that UPI interface is ready.
                                ; If not, loop until it is.

        rbit    gie,enir        ; * INDIVISIBLE SEQUENCE *
        rbit    austat,alerth.b ; Clear ALERT bit.
        ld      obuf,ustat      ; Load UART Status Byte into OBUF for CPU.
        rbit    brkflg,ustat    ; Clear END OF BREAK indication.
        sbit    gie,enir        ; * END INDIVISIBLE SEQUENCE *
        ret                     ; Return to main loop.

        .form   'Main:  Report a Data Error Condition to CPU'

snderr:                         ; Send DATA-ERR interrupt to CPU.
        rbit    aerr,alerth.b   ; Clear ALERT bit.
        jsrl    rdwait          ; Check that UPI interface is ready.
                                ; If not, wait until it is.

        ld      obuf,#verr      ; Load DATA-ERR vector into OBUF for CPU.
        jsrl    rdwait          ; Check that UPI interface is ready.
                                ; If not, wait until it is.

        ld      obuf,errchr     ; Give CPU the offending character.
        jsrl    rdwait          ; Check that UPI interface is ready.
                                ; If not, wait until it is.

        ld      obuf,errfgs     ; Give CPU the error flags.
        ret                     ; Return to main program loop.

        .form   'Main:  Send UART Acknowledge interrupt to CPU'

snduak:                         ; Send ACK-UART interrupt to CPU.
        rbit    auack,alerth.b  ; Clear ALERT bit.
        jsrl    rdwait          ; Check that UPI interface is ready.
                                ; If not, loop until it is.

        ld      obuf,#vuack     ; Load ACK-UART vector into OBUF for CPU.
        ret                     ; Return to main program loop.


        .form   'Main:  Send Diagnostic Interrupt to CPU'

sndiag:
        jsrl    rdwait          ; Wait for UPI interface ready.
        ld      obuf,#vdiag     ; Load vector into OBUF for CPU.
        jsrl    rdwait          ; Wait for UPI interface ready.
        rbit    gie,enir        ; *** Begin Indivisible Sequence ***
        ld      obuf,dsevc      ; Transfer Severity Code.
        ld      dsevc,#0        ; Clear it.
        ld      A,derrc         ; Get Error Code.
        ld      derrc,#0        ; Clear it.
        rbit    adiag,alerth.b  ; Clear ALERT bit.
        sbit    gie,enir        ; *** End Indivisible Sequence ***
```

TL/DD/9977–30

```
        jsrl    rdwait          ; Wait for UPI interface ready.
        st      A,obuf          ; Transfer Error Code.
        jsrl    rdwait          ; Wait for UPI interface ready.
                        ; Remaining bytes will have meaning only for
                        ;  command errors.
        ld      obuf,dbyte      ; Transfer Byte Received.
        jsrl    rdwait          ; Wait for UPI interface ready.
        ld      obuf,dccmd      ; Transfer Current Command.
        jsrl    rdwait          ; Wait for UPI interface ready.
        ld      obuf,dqual      ; Transfer Command Count.
        ret                     ; Return to main program loop.

        .form   'UPI (I3) Interrupt:  Data from CPU'

        .ipt    3,upiwr         ; Declare upiwr as vector for Interrupt 3.

upiwr:                          ; Write Strobe received from CPU.
        push    A               ; Save Context
        push    psw

        ld      upicsv.b,upic   ; Save UPIC register image for LA0 bit test.

        ifbit   cmdemp,curcmd   ; If expecting first byte of a command,
        jmpl    firstc          ;  then go process it as such.

        ld      A,ibuf          ; If not, input it for entry into cpubuf.

        ifeq    A,#x'A5         ; Check for RESET command.
        jp      lcrst
        ifbit   la0,upicsv.b    ; Check for command argument written to proper
                                ;  address.
        jp      lcord           ; If so, go process as a normal argument.
        jsrl    hangup          ; If not, process as a FATAL error, generating
                                ;  !DIAG interrupt.

lcrst:  ifbit   la0,upic        ; Continue checking for a RESET command.
        jp      lcord
        jmpl    xreset          ; If so, go reset the HPC.

lcord:  x       A,[cpuad].b     ; If not, place it in next available cpubuf
                                ;  entry.
        inc     cpuad
        decsz   numexp
        jmpl    upwret          ; If not final byte of command, then return.

lastc:  ld      A,curcmd        ; Else, process current command.
        ifbit   getcnt,A.b      ; Check if extended collection is being made.
        jp      lastcl          ; If not, then:
        sbit    cmdemp,curcmd   ;   Set command slot available again.
        ld      cpuad,#cpubuf   ;   Reset CPU buffer pointer to beginning.

lastcl:         and     A,#x'1F         ; Mask off flag bits.
        shl     A               ; Scale by two, and then
        .odd
        jidw                    ;  jump based on command value.
        .ptw    lcinit,lcselc,lcselu,illc
        .ptw    illc,illc,illc,illc  ; (All these are one-byte commands.)
        .ptw    lcscst,lcslcv,lcslcd,lcsled
        .ptw    illc,lcsndu,illc,illc
        .ptw    illc,illc
```

TL/DD/9977–31

```
                        ; Process INITIALIZE Command.

lcinit:         ld      rtevs,#x'01     ; Enable only Real-Time Clock interrupts, but
        ifeq    cpubuf.b,#0     ;  disable them again if
        rbit    rtcenb,rtevs    ;  the command argument is zero.
        ld      rtcivl,cpubuf.b ; Put argument into Real-Time
                                ;  Clock interval.
        ld      rtccnt,cpubuf.b ; Put argument into Real-Time
                                ;  Clock count.
        sbit    tltie,tmmdl     ; Enable Timer T1 interrupt, if not already
                                ;  enabled.
        rbit    tlstp,tmmdl     ; Start timer, if not already running.

        jsrl    lcibuf          ; Initialize buffer parameters.
        ld      alert.w,#0      ; Set no events pending.
        ld      ackmd,#1        ; BUSY will fall during ACK/ pulse.
        ld      errchr,#55      ; Arbitrary fill for error character.
        ld      errfgs,#0       ; Clear error detail flags.
        ld      swlast,#0       ; Set up initial switch values.
        ld      swlsnt,#0       ; (Both current and last sent)


                                ; Reset Centronics port:  Busy

incent:         ld      cps,#x'25       ; Initialize Centronics port status byte
                                ;  in memory.  (Busy, and PRIME interrupt
                                ;  disabled;  otherwise normal.)
        jsrl    setcen          ; Send to Centronics Control Latch.

                                ; Reset UART port:  Busy

inuart:         and     enui,#x'FC      ; Disable UART by clearing enables on
                                ;   UART-generated interrupts (except EXUI/,
                                ;   which is connected to INPUT PRIME/.)
        ld      ups,#x'03       ; Flag UART as busy and not selected.
        ld      A,rbuf          ; Clear out spurious characters.
        ld      A,enur          ; Clear out spurious error flags.
        jmpl    upwret          ; Return.

lcibuf:                         ; Internal subroutine to initialize buffer status.
                ;  Called also from SELECT commands.
        ld      numchr,#0       ; Clear count of characters received.
        ld      cadin,#botad    ; Next character in from comm port goes to
                                ;  first byte of buffer.
        ld      cadout,#botad   ; Next port data character out (to CPU)
                                ;  comes from first byte of buffer.
        ld      numout,#0       ; No characters being sent to CPU.
        ld      cntout,#0       ; No characters being sent to CPU.
        ld      bstat,#0        ; Set buffer ready to receive.
        and     lcvs,#x'0F      ; (DEBUG:  Initialize LCV latch high bits.)
        ld      portah,lcvs
        sbit    lcvclk,portbh
        rbit    lcvclk,portbh
        ret                     ; Return.


                        ; Process SELECT-CENT command.

lcselc:         and     enui,#x'FC      ; Disable UART by clearing enables on
```

TL/DD/9977–32

```
                              ;   UART-generated interrupts (except EXUI/,
                              ;   which is connected to INPUT PRIME/.)
        rbit    usel,ups      ; Flag UART not selected.
        ifbit   flemp,uflow   ; If valid UART mode exists,
        jp      lcsecl
        jsrl    dtroff        ;   use it to set DTR to "not ready" state.

lcsecl: ld      ackmd,cpubuf.b ; Accept ACK/ mode from command buffer.
        ld      pascnt,cpubuf+1.b     ; Put "Buffer Pass" value into
                                      ;  the PASCNT slot.
        ld      stpcnt,cpubuf+2.b     ; Put "Host Stop" value into
                                      ;  the STPCNT slot.
        jsrl    lcibuf        ; Initialize buffer parameters.

primlp: ifbit   uart,irpd     ; Check to see if INPUT PRIME/ interrupt is
        jp      primlp        ;   still asserted.  If so, wait here.

        sbit    i4,ircd       ; Set up STROBE detector to see leading edge.
        ld      irpd,#x'EF    ; Clear any spurious interrupt triggered by
                              ;  polarity change.
        sbit    i4,enir       ; Enable interrupts on I4 (STROBE).
        sbit    uart,enir     ; Enable INPUT PRIME/ interrupt (through
                              ;   UART vector).
        ld      cps,#x'A9     ; Set Centronics interface byte not busy,
                              ;   selected, and all status bits normal.
        jsrl    setcen        ; Clears BUSY signal and generates ACK/ pulse
                              ;  according to current mode in ACKMD.
        jmpl    upwret        ; Return.


                ; Process SELECT-UART command.

lcselu: ld      A,divby.b     ; Process UART baud selection.
        and     A,#x'0F       ; Strip out old baud rate selector.
        st      A,cpubuf+7.b  ; Save (in unused area of the command buffer),
                              ;  and start processing new value.
        ifgt    cpubuf.b,#x'08  ; Check if out of range.
        jsrl    hangup
        ld      A,#10
        sc
        subc    A,cpubuf.b    ; Convert to DIVBY field format.
        swap    A             ; Place value in correct field.
        or      A,cpubuf+7.b  ; OR with Microwire rate field.
        st      A,divby.b     ; Place back in DIVBY register.

        ld      uframe,cpubuf+1.b ; Get requested frame format.
        and     uframe,#x'07  ; Discard unused bits.
        sbit    b8or9,enu     ; Set 9-bit mode for 8-bit data plus parity.
        ifgt    uframe,#1     ; If 7-bit plus parity, or 8-bit without parity,
        rbit    b8or9,enu     ;  then change this setting to 8-bit mode.

        rbit    b2stp,enui    ; Initialize to one Stop bit.
        ifeq    uframe,#3     ; Test for number of Stop bits requested,
        sbit    b2stp,enui    ;  and set up UART hardware accordingly.
        ifgt    uframe,#5
        sbit    b2stp,enui

        ld      A,cpubuf+2.b  ; Set up handshaking mode.  This also clears
        and     A,#x'0F       ;  the FLEMP bit automatically.
        st      A,uflow
```

TL/DD/9977–33

```
        ld      pascnt,cpubuf+3.b       ; Put "Buffer Pass" value into
                                        ;  the PASCNT slot.
        ld      stpcnt,cpubuf+4.b       ; Put "Host Stop" value into
                                        ;  the STPCNT slot.
        jsrl    lcibuf          ; Initialize buffer parameters.

        ld      cps,#x'25       ; Set up Port A to disable and de-select
                                ;  Centronics port, and disable
                                ;  INPUT PRIME interrupt.
        rbit    clinmd,ackmd    ; Clear the Centronics Line Mode bit.
        jsrl    setcen          ; Send to Centronics latch and to Busy flag.
        rbit    i4,enir         ; Disable Centronics STROBE interrupt.
        ld      A,rbuf          ; Clear any pending character before selection.
        ld      A,enur          ; Clear any error indications before selection.
        sbit    eri,enui        ; Enable receiver interrupt.
        rbit    eti,enui        ; Disable transmitter interrupt.
        ld      ups,#x'80       ; Set UART port selected, not busy, and
                                ;  no characters being sent or waiting to be
                                ;  sent.
        ld      ustat,#x'01     ; Set DSR ready(will trigger interrupt if not).
        sbit    uart,enir       ; Enable UART interrupt.

        ifbit   dtrb0,uflow     ; Initialize DTR pin according to new mode.
        jp      lcslul
        rbit    dtr,portbl
        jp      lcslu2
lcslul: sbit    dtr,portbl
lcslu2:

        jmpl    upwret          ; Return.



                ; Process SET-CENT-STS Command.

lcscst: ld      cps,cpubuf.b    ; Load Centronics Port Status from byte
                ;   provided by CPU.
        jsrl    setcen          ; Perform ACK/ if new status calls for it.
        jmpl    upwret


                ; Process SET-CONTRAST Command.

lcslcv: ld      A,cpubuf.b      ; Load LCD Voltage latch (Contrast) from byte
                ;   supplied by CPU.
        comp    A               ; (3-bit value is in complemented form.)
        and     A,#x'07         ; Use only lower three bits.
        and     lcvs,#x'F8      ; Clear field in memory image.
        or      lcvs,A.b        ; Merge new field into image.
        ld      portah,lcvs     ; Place on Port A (input to latch).
        sbit    lcvclk,portbh   ; Clock latch.
        rbit    lcvclk,portbh
        jmpl    upwret


                ; Process SEND-LCD Command.

lcslcd: ifbit   getcnt,curcmd           ; Check for first or second collection
        jmpl    lcslcl                  ;  phase.
```

TL/DD/9977–34

```
lcslc2:                                 ; Second phase:  begins execution of the LCI
                                ;   command.
        ld      lcdbuf.w,cpubuf.w       ; Copy CPU buffer to LCD string buffer.
        ld      lcdbuf+2.w,cpubuf+2.w
        ld      lcdbuf+4.w,cpubuf+4.w
        ld      lcdbuf+6.w,cpubuf+6.w
        ld      lcdsct,lcdnum           ; Move number of characters to string
                                        ;  count byte
        inc     lcdsct                  ;  (incremented by one because of
                                        ;   extra interrupt occurring after
                                        ;   last character has been sent).
        ld      lcdsix,#lcdbuf          ; Set string pointer to first byte.
        ld      lcdsfg,lcdfgs           ; Move flag bits to string location.

        ld      r6,#x'FFFF              ; Set up R6 and T6 to trigger string
        ld      t6,#0                   ;  transfer.
        sbit    t6tie,pwmdh             ; Enable timer T6 interrupt.
        rbit    t6stp,pwmdh             ; Start timer to trigger (immediate)
                                        ;  interrupt from timer T6.
        jmpl    upwret

lcslc1:                                 ; First phase:  Prepare to collect up to 8
                                ;   more bytes of command.
        ld      lcdfgs,cpubuf.b         ; Get flag bits supplied by CPU.
        ld      lcdnum,cpubuf+1.b       ; Get character count from CPU.

        ld      numexp,lcdnum           ; Request another collection of
                                        ;  data from the CPU (the string of
                                        ;  data for the panel).
        ld      cpuad,#cpubuf           ; Reset CPU collection pointer to start
                                        ;  of command buffer.
        rbit    getcnt,curcmd           ; Declare that it will be the final
                                        ;  collection.
        jmpl    upwret


                ; Process SEND-LED Command.

lcsled:         ld      A,cpubuf.b      ; Load LED latch from byte supplied by CPU.
        comp    A               ; (Data goes to LED's in complemented form.)
        st      A,portah        ; Place new value on Port A (input to latch).
        sbit    ledclk,portbh   ; Clock latch.
        rbit    ledclk,portbh
        jmpl    upwret


                ; Process SEND-UART Command.

lcsndu:
        ld      uschr,cpubuf.b ; Queue this character,
        sbit    schr,ups        ;  and request transmission at next
                                ;  transmitter interrupt.
        ifbit   eti,enui        ;  Check to see if another character is
        jmpl    upwret          ;  already being sent (transmitter interrupt
                                ;  enabled).
        jsrl    setuar          ;  If not, then call flow control routine to
                                ;    send it.
        jmpl    upwret          ; Return.
```

TL/DD/9977–35

```
        .form   'Processing of First Byte of Command (Code)'

                ;  One-byte commands are processed in this section.
                ;  Longer commands are scheduled for collection of
                ;    remaining bytes, and are processed in routines
                ;    above.

firstc:         ld      A,ibuf          ; Get command from UPI port.
        ifbit   la0,upicsv.b    ; Check for out-of-sequence condition
                                ;  (argument instead of command).
        jsrl    hangup          ; If so, process as a FATAL error (previous
                                ;  command was too short).

                        ; Processing of RESET command.

        ifeq    A,#x'A5         ; Check for RESET command.
        jp      xreset
        jp      fcord

                                ; This code is entered whenever a RESET
                                ;  command is received.
xreset:
        ld      obuf,#vdiag     ; Present dummy value for CPU,
                                ;  (in case a value was already in OBUF),
        jsrl    rdwait          ;  and wait for it to be read by CPU.
        ld      A,#0            ; Initialize registers.
        st      A,upic.b
        st      A,ibuf.w        ; (Actually all of DIRA.)
        st      A,dirb.w
        st      A,bfun.w
        st      A,ircd.b
        st      A,portp.w
        st      A,sp.w          ; Then, through RESET vector,
        st      A,psw.w
        ret                     ;  jump to start of program.

                        ; Here, process an ordinary command (not RESET).

fcord:
        and     A,#x'1F         ; Use only least-significant 5 bits.
        ifgt    A,#x'11         ; Check for command out of range.
        jmpl    illc
        st      A,curcmd        ; Save as current command.

        shl     A               ; Scale by two, and then
        .odd
        jidw                    ;  jump based on command value.
        .ptw    fcinit,fcselc,fcselu,illc
        .ptw    fcflsh,fccbsy,fccnby,fcifby
        .ptw    fcscst,fcslcv,fcslcd,fcsled
        .ptw    fcbeep,fcsndu,fcusts,illc
        .ptw    illc,illc

fcinit:         ld      numexp,#1       ; First byte of INITIALIZE command.
                                ;  Expects 1 more byte (RTC interval).
        jmpl    upwret          ; Return.

fcselc:         ld      numexp,#3       ; First byte of SELECT-CENTRONICS command.
```

TL/DD/9977–36

```
                              ;  Expects 3 more bytes (ACK-Mode, Pass-Count,
                              ;  Stop-Count).
        jmpl    upwret        ; Return.

fcselu:         ld      numexp,#5       ; First byte of SELECT-UART command.
                              ;  Expects 5 more bytes (baud, frame,
                              ;  handshake, Pass-Count, Stop-Count)
        jmpl    upwret        ; Return.

                        ; Processing of one-byte FLUSH-BUF command.
fcflsh:         sbit    aflush,alert.b  ; Set flush request bit in ALERT byte.
        sbit    cmdemp,curcmd   ; Set command byte empty (end of command).
        jmpl    upwret

                        ; Processing of one-byte CPU-BUSY command.
fccbsy:         sbit    cpubsy,bstat    ; Set CPU Busy bit in BSTAT byte.
        sbit    6,lcvs          ; (DEBUG:  set also CPU Busy bit in LCV latch.)
        ld      portah,lcvs
        sbit    lcvclk,portbh
        rbit    lcvclk,portbh
        sbit    cmdemp,curcmd   ; Set command byte empty (end of command).
        jmpl    upwret

                        ; Processing of one-byte CPU-NOT-BUSY command.
fccnby:         rbit    cpubsy,bstat    ; Reset CPU Busy bit in BSTAT byte.
        rbit    6,lcvs          ;(DEBUG: reset also CPU Busy bit in LCV latch.)
        ld      portah,lcvs
        sbit    lcvclk,portbh
        rbit    lcvclk,portbh
        sbit    cmdemp,curcmd   ; Set command byte empty (end of command).
        jmpl    upwret

fcifby:                         ; Processing of one-byte SET-IFC-BUSY command.
                        ; This command (one byte) sets the interface busy
                        ;   immediately, to stop characters from the external
                        ;   system.
        sbit    cmdemp,curcmd   ; Set command byte empty (end of command).
        ifbit   usel,ups        ; Check if UART is selected.
        jmpl    fcibyu          ; If so, go set up flow control.
        ifbit   enprm,cps       ; Check if Centronics port is selected.
        jmpl    fcibyc          ; If so, go set up Centronics BUSY status.
        jsrl    hangup          ; Otherwise, error.  Stop.

fcibyu:                                 ; Set UART port busy.
        sbit    cus,ups         ; Set UART input port status busy.
        jsrl    dtroff          ; Set DTR handshake appropriately.
        ifbit   eti,enui        ; Check if UART transmitter busy.
        jp      fcibyl          ;   If so, flow control will happen
                                ;     automatically.
        ifbit   xonb,uflow      ;   If not, then if XON mode is selected,
        jsrl    setuar          ;     invoke flow control routine.
fcibyl:         jmpl    upwret

                        ; Set Centronics port busy.
fcibyc:         sbit    ifcbsy,bstat    ; Set Interface Busy bit in BSTAT byte.
        sbit    cbusy,cps       ; Set BUSY bit in Centronics Port Status byte.
        jsrl    setcen          ; Change Centronics port control latch
                                ;   accordingly.
        sbit    cmdemp,curcmd   ; Set command byte empty (end of command).
        jmpl    upwret
```

TL/DD/9977–37

```
                                ; First byte of SET-CENT-STS command.
fcscst:         ld      numexp,#1       ; Set up to expect one more byte.
        jmpl    upwret

                                ; First byte of SET-CONTRAST command.
fcslcv:         ld      numexp,#1       ; Set up to expect one more byte.
        jmpl    upwret

                                ; First byte of SEND-LCD command.
fcslcd:         ld      numexp,#2       ; Set up to expect one more byte.
        sbit    getcnt,curcmd   ; Note extended collection mode in Current
                                ;  Command byte.
        jmpl    upwret

                                ; First byte of SEND-LED command.
fcsled:         ld      numexp,#1       ; Send to LED's: Set up to expect one more byte.
        jmpl    upwret

                                ; Process one-byte BEEP command.
fcbeep:         sbit    cmdemp,curcmd   ; No arguments; set CURCMD byte empty.
        sbit    t7tfn,portph    ; Enable beep tone to panel speaker.
        sbit    t0tie,tmmdl     ; Enable Timer T0 interrupt.
        ld      beepct,#19      ; Initialize duration count (approximately
                                ;  1 second, in units of Timer T0 overflows).
        jmpl    upwret

                                ; First byte of SEND-UART command.
fcsndu:         ld      numexp,#1       ; Send to UART:  Set up to expect one more byte.
        jmpl    upwret

                                ; Process one-byte TEST-UART command.
fcusts:         sbit    cmdemp,curcmd   ; No arguments;  set CURCMD byte empty.
        sbit    austat,alerth.b ; Force UART Status interrupt.
        jmpl    upwret


illc:   jsrl    hangup          ; Process illegal command codes.


                                ; Return from UPI Write interrupt.
upwret:                                 ; Restore Context
        pop     psw
        pop     A
        reti

        .form   'Timer Interrupt Handler'

        .ipt    5,tmrint        ; Declare entry point for Timer Interrupt.

tmrint:         push    A               ; Save context.
        push    B               ;
        push    psw             ;

tlpoll:         ifbit   tlpnd,tmmdl     ; Poll for Timer T1 interrupt (Real-Time Clock).
        jmpl    tlint           ; If set, go service it.

t6poll:         ifbit   t6pnd,pwmdh     ; Poll for Timer T6 interrupt (LCD Panel Timing
        jmpl    t6int           ;  Interrupt).
```

TL/DD/9977–38

```
t0poll:         ifbit   t0pnd,tmmdl     ; Poll for Timer T0 interrupt (Beep Duration).
        jp      t0pdg           ; If set, check the Enable bit;  T0 is not
        jp      t0notp          ;  always enabled to interrupt when it runs.
t0pdg:  ifbit   t0tie,tmmdl     ; If enable is also set, then go service T0.
        jmpl    t0int
t0notp:                                 ; (This label is deliberately here.)

noint:  jsrl    hangup          ; Error:  no legal timer interrupt pending.

        .form   'Timer T1 Interrupt Service Routine'

tlint:  sbit    tlack,tmmdl     ; Acknowledge T1 interrupt.
        ifbit   rtcenb,rtevs    ; Check if RTC interrupts are enabled.
        jp      tlintl
        jmpl    kbdchk          ; If not, then go check other events.
tlintl:         decsz   rtccnt          ; Decrement interval value.
        jmpl    kbdchk          ; If interval has not elapsed, then go check
                                ;  for other events.
        ld      rtccnt,rtcivl   ; Reload counter value for next interval.
        ifbit   artc,alert.b    ; Check if CPU has received previous interrupt
        jp      tlrerr          ;  request;  report error if not.
        sbit    artc,alert.b    ; Set Real-Time Interrupt request to main
        jp      kbdchk          ;  program.
tlrerr:         sbit    0,dsevc         ; Signal NOTE severity.
        sbit    7,derrc         ; Signal multiple-RTC error.
        sbit    adiag,alerth.b  ; Request !DIAG interrupt from main program.

kbdchk:                                 ; Check keyboard switches.
        rbit    astts,portbh    ; Enable pushbutton data to Port D.
        ld      A,portd         ; Sample pushbutton switches.
        sbit    astts,portbh    ; Disable pushbutton data to Port D.
        xor     A,#x'FF         ; Complement low-order 8 bits of A.
        x       A,swlast        ; Exchange with last sample.
        ifeq    A,swlast        ; Check if the data is stable (same as last
                                ;  sample).
        jp      kbintl
        jmpl    dsrchk          ; If not, go check other events.

kbintl:         ifeq    A,swlsnt        ; Check if the data differs from the last
                                ;  pattern sent to the CPU.
        jmpl    dsrchk          ; If not, go check other events.

        st      A,swlsnt        ; Place new pattern in "last sent" location.
        sbit    abutton,alerth.b ; Request "BUTTON-DATA" interrupt to CPU.

dsrchk:                         ; Check for status of DSR signal if mode selected.
        ifbit   usel,ups        ; Check if UART is selected.
        jp      dsr0
        jmpl    tmochk          ; If not, skip both DSR and BREAK checking.
dsr0:   ifbit   dsrb,uflow      ; Check if DSR input should be checked.
        jp      dsrl
        jmpl    brkchk
dsrl:   ld      A,#x'01         ; Initialize Accumulator to check DSR.
        ifbit   dsr,porti       ; Check current state of DSR pin.
        rbit    0,A             ; Clear LSB of A if DSR pin set.
        st      A,B             ; Register B holds DSR state (1 = DSR Ready).
        ifbit   dsrflg,ustat    ; Check last DSR state given to CPU.
        xor     A,#x'01         ; Toggle LSB of A if set.
        ifbit   0,A             ; If LSB of A is still set, then must send
        jp      dsr2            ;  UART-STATUS interrupt to CPU.
```

TL/DD/9977–39

```
        jmpl    brkchk          ; Else, go check BREAK status.
dsr2:   rbit    dsrflg,ustat    ; Report new state of DSR to CPU.
        ifbit   0,B.b
        sbit    dsrflg,ustat
        sbit    austat,alerth.b ; Request main program to generate !UART-STATUS.

        ifbit   0,B.b           ; Now, enable or disable UART receiver based on
        jp      dsron           ;  new DSR state.
dsroff:         rbit    eri,enui        ; If DSR is now inactive, disable receiver
        jmpl    brkchk          ;  interrupts.
dsron:  ld      A,ups           ; If DSR is now active, check to see whether
        and     A,#x'60         ;  receiver may be re-enabled:  must test
        ifgt    A,#x'00         ;  for BREAK condition and Multiple Character
        jmpl    brkchk          ;  Error condition, which disable the receiver
        sbit    eri,enui        ;  until a SELECT-UART command.  If not
                                ;  permanently disabled then re-enable it here.
        ld      A,rbuf          ;  Also remove any garbage characters and error
        ld      A,enur          ;  indications seen while DSR was inactive.

brkchk:         ifbit   brkmd,ups       ; Check whether BREAK has been detected.
        jp      brkmdl
        jmpl    tmochk          ; Go check for other events if not.
brkmdl:         ifbit   txd,portbl      ; Check UART data input pin.
        jp      brkmd2          ; If set, BREAK pulse is done.
        jmpl    tmochk          ; Otherwise, go check for other events.
brkmd2:         rbit    brkmd,ups       ; Clear BREAK mode in UART Port Status byte.
        sbit    brkflg,ustat    ; Set END OF BREAK bit in UART status to CPU.
        sbit    austat,alerth.b ; Request main program to generate !UART-STATUS.

tmochk:                 ; *** Insert other RTC events here. ***

        jmpl    tmrret          ; Return from Timer T1 interrupt.

        .form   'Timer T6 Interrupt Service Routine'

                        ; Timer T6 interrupt routine:  sends characters from
                        ;  LCD String Buffer to the panel.
t6int:  sbit    t6stp,pwmdh     ; Stop timer T6.
        sbit    t6ack,pwmdh     ; Acknowledge T6 interrupt.

        decsz   lcdsct          ; Decrement LCD character count.
        jmpl    t6nxtc          ; If not done, go send another character.

        sbit    alcdak,alert.b  ; If done, request main program to send LCD
                                ;  Acknowledge interrupt to CPU.
        jmpl    tmrret

t6nxtc:         ld      A,lcdsfg        ; Get flags byte (for panel RS signal).
        shr     A               ; Shift right, LSB into carry.
        st      A,lcdsfg        ; Store shifted value back.
        sbit    pnlrs,lcvs      ; Determine proper state for RS signal from
        ifc                     ;  current character's flag (= flag inverted).
        rbit    pnlrs,lcvs
        ld      portah,lcvs     ; Send new RS value to LCD Voltage (LCV) latch.
        sbit    lcvclk,portbh   ; Clock the latch.  RS signal is now valid.
        rbit    lcvclk,portbh

        ld      A,[lcdsix].b    ; Get next LCD character from string buffer.
        inc     lcdsix          ; Increment character pointer.
        comp    A               ; Complement character, then
```

TL/DD/9977–40

```
        st      A,portah        ;  place it on Port A for LCD display.
        rbit    pnlclk,portbl   ; Clock it into panel.
        sbit    pnlclk,portbl
        comp    A               ; Restore A to uncomplemented form for
                                ;  test performed below.

        ld      t6,#148         ; Set up normal delay time in timer T6
                                ;  (120 microseconds).
        ifgt    A,#x'03         ; Check whether the longer delay
        jp      t6nxt2          ;  (4.9 milliseconds) is necessary.
                                ;  This happens if RS=0 and the byte sent to
        ifnc                    ;  the panel is a value of hex 03 or less.
        ld      t6,#6022        ; If so, change timer to 4.9 milliseconds.

t6nxt2:         rbit    t6stp,pwmdh     ; Start Timer T6 to time out the character.
        jmpl    tmrret          ; Return from the interrupt.

        .form   'Timer T0 Interrupt Service Routine'

t0int:                  ; Count duration of beep tone.  Restore beep signal
                        ;  to zero and re-enable switch sampling interrupt
                        ;  when done.
        sbit    t0ack,tmmdl     ; Acknowledge interrupt from Timer T0.
        decsz   beepct          ; Check whether beep time has finished.
        jmpl    tmrret          ;  No:  return from interrupt.
        rbit    t0tie,tmmdl     ;  Yes:  disable Timer T0 interrupts and
                                ;        continue.
        and     portph,#x'0F    ; Disable speaker output.
        jmpl    tmrret          ; Return from interrupt.

                        ; Common return for timer interrupt service routines.
tmrret:         pop     psw             ; Restore context.
        pop     B
        pop     A
        reti

        .form   'Centronics Port Interrupt Handler'
;
;       Centronics Port Interrupt Handler
;               (Pin I4 rising edge)
;
;       Note that cadin is an 8-bit quantity;  buffer must be
;               contiguous within the basepage area.
;

        .ipt    4,cenint

cenint:         push    psw             ; Save context.
        push    A
        push    B
        push    K

                ; Decide whether to process leading or trailing edge interrupt.
        ifbit   i4,ircd         ; Check polarity of detector.
        jmpl    cstrbl          ; Leading edge (rising on I4 pin).
        jmpl    cstrbt          ; Trailing edge (falling on I4 pin).


cstrbl:                         ; STROBE/ leading edge service routine.
```

TL/DD/9977–41

```
        ld      K,#topad        ; Reg. K gets buffer top address.
        sbit    astts,portbh    ; Make sure pushbutton buffer is off.
        rbit    cdata,portbh    ; Enable Centronics data to Port D.

                        ; Test whether there is room for another byte
                        ;  in the data buffer.
        ifbit   full,bstat      ; If FULL bit set,
        jmpl    cenerr          ;  process this character as an error
                                ;  (Buffer Overflow).

        ld      B,cadin         ; Get current buffer input address.
        ld      A,portd         ; Get character.
        xs      A,[B+].b        ; Store in table.
        jp      cen0            ; If skip,
        ld      B,#botad        ;   then wrap input pointer to beginning
cen0:   ld      cadin,bl.b      ;   of buffer;  else just increment it.

cen1:   inc     numchr          ; Increment number of characters.
        ifgt    pascnt,numchr   ; Check if buffer full enough to send.
        jmpl    cenlex          ;     No:  end of service.

        sbit    pass,bstat      ;     Yes: indicate buffer ready to pass.
        sbit    4,lcvs          ; (DEBUG:  report status in LCD Contrast latch.)
        ld      portah,lcvs
        sbit    lcvclk,portbh
        rbit    lcvclk,portbh

        ifgt    stpcnt,numchr   ; Check if buffer too full for more
                                ;     host characters.
        jmpl    cenlex          ;  No:  end of service.

        sbit    cbusy,cps       ;  Yes:  set Centronics port status busy.
        sbit    stop,bstat      ;        set Buffer Status as "STOPPED".
        sbit    5,lcvs          ; (DEBUG: report status in LCD Contrast latch.)
        ld      portah,lcvs
        sbit    lcvclk,portbh
        rbit    lcvclk,portbh

        ifeq    numchr,#bufsiz  ; Check if buffer completely full.
        sbit    full,bstat      ;  Yes:  set condition.

        jmpl    cenlex          ;        Update Centronics latch and quit.

cenerr:                         ; Error handler:  invoked if BUSY flag fails to stop
                        ;  host processor and the HPC's data buffer overflows
                        ;  as a result.
        sbit    cbusy,cps       ; Set busy indication in Centronics Port
                                ;   Status byte (to keep BUSY asserted to host
                                ;   when ENCDATA/ signal is removed later).
                                ;   This should not be necessary except in case
                                ;   of an internal error in this program.

        sbit    7,lcvs          ; (DEBUG:  report error in LCD Contrast latch.)
        ld      portah,lcvs
        sbit    lcvclk,portbh
        rbit    lcvclk,portbh

        ifbit   aerr,alerth.b   ; If an error has already been posted,
        jp      cenmer          ;   handle as a multiple error.
        jmpl    cenler          ; Else, report single error.
```

TL/DD/9977–42

```
cenmer:         sbit    bufovf,errfgs   ; OR in the buffer overflow condition.
        sbit    errovf,errfgs   ; Update error conditions byte to also report
                                ;  an error overflow.
        rbit    i4,enir         ; Disable STROBE interrupt until re-initialized
                                ;  by CPU.
        jmpl    cenlex          ; Return from the interrupt.

cenler:         sbit    aerr,alerth.b   ; Signal an error.
        ld      errfgs,#x'10    ; Report buffer overflow as reason.
        ld      errchr,portd    ; Place character in ERRCHR slot for report to
                                ;   CPU.
        ld      fshlim,numchr   ; Establish limit on future flushes.
        jmpl    cenlex          ; Return from the interrupt.


cenlex:                         ; Exit from Centronics STROBE/ leading edge.
        ld      A,cps           ; Prepare to keep BUSY active when ENCDATA/
        sbit    cbusy,A.b       ;  is removed.
        st      A,portah        ; Send CPS byte (with BUSY set) to Centronics
                                ;  status latch.
        sbit    cenclk,portph   ;  (Pulse latch strobe.)
        rbit    cenclk,portph
        sbit    cdata,portbh    ; Remove Centronics data enable;  loads BUSY
                                ;  signal with a "1".
        rbit    i4,ircd         ; Set I4 strobe pin to trigger on STROBE/
                                ;  trailing edge.
        ifbit   i4,porti        ; Check if strobe has already gone away.
        jmpl    cenend          ;  If not, just return (no ACK/ pulse).
                                ;   The "cstrbt" routine will be activated then
                                ;   whenever STROBE/ goes away, by means of the
                                ;   I4 interrupt.
        jmpl    cstrbt          ;  If so, there is a very small possibility
                                ;   that the interrupt request may have been
                                ;   lost due to it changing while the polarity
                                ;   bit in IRCD was being changed above.
                                ;   Jump to trailing edge service routine
                                ;   directly from here.


cstrbt:                         ; Centronics STROBE/ trailing edge.

        sbit    i4,ircd         ; Set up for leading edge detection again.
        ld      irpd,#x'EF      ; Clear interrupt I4, in case the leading edge
                                ;  routine came directly here.  (No hardware
                                ;  clear of the request occurs in that case.)
        jmpl    cenupd          ; Go update Centronics port, with ACK/ pulse
                                ;  if necessary.

;       Return from interrupt.

        ; With Centronics Port update.
cenupd:         jsrl    setcen          ; Update Centronics Control signals
                                ;  from CPS byte.

        ; Without Centronics Port update.
cenend:         pop     K               ; Restore context from stack and return from
                                ;  Centronics interrupt.
        pop     B
        pop     A
```

TL/DD/9977–43

```
        pop     psw
        reti                    ; Return from Centronics interrupt.


;       Subroutine SETCEN.
;       Sets up Centronics Port control signals according to CPS byte.
;       Generates ACK signal (if called for) according to current
;               Centronics timing mode (in ACKMD byte).
;       Trashes Accumulator.

setcen:         rbit    cdata,portbh    ; Start with ENCDATA/ low, regardless
                                        ;  of previous state.

        ifbit   cbusy,cps       ; Check if BUSY flag should stay set.
        jmpl    noack           ; If so, no ACK/ pulse.

        ld      A,ackmd         ; Get ACK/ mode,
        and     A,#x'03         ;  and extract the timing field.
        jid                     ; Branch based on ACK/ timing mode.
        .pt     aab,aba,baa

aab:    ld      portah,cps      ; BUSY low after ACK/ pulse.
        rbit    cack,portah     ;  ACK/ falling edge.
        sbit    cenclk,portph   ;       Pulse CCTLCLK to load latch.
        rbit    cenclk,portph
        sbit    cack,portah     ;  ACK/ rising edge.
        sbit    cenclk,portph   ;       Pulse CCTLCLK to load latch.
        rbit    cenclk,portph
        sbit    cdata,portbh    ; Load BUSY flag.
        ret

aba:    ld      portah,cps      ; BUSY low during ACK/ pulse.
        rbit    cack,portah     ;  ACK/ falling edge.
        sbit    cenclk,portph   ;       Pulse CCTLCLK to load latch.
        rbit    cenclk,portph
        sbit    cdata,portbh    ; Load BUSY flag.
        sbit    cack,portah     ;  ACK/ rising edge.
        sbit    cenclk,portph   ;       Pulse CCTLCLK to load latch.
        rbit    cenclk,portph
        ret

baa:    ld      portah,cps      ; BUSY low before ACK/ pulse.
        sbit    cdata,portbh    ; Load BUSY flag.
        rbit    cack,portah     ;  ACK/ falling edge.
        sbit    cenclk,portph   ;       Pulse CCTLCLK to load latch.
        rbit    cenclk,portph
        sbit    cack,portah     ;  ACK/ rising edge.
        sbit    cenclk,portph   ;       Pulse CCTLCLK to load latch.
        rbit    cenclk,portph
        ret

noack:  ld      portah,cps      ; BUSY high:  Set Centronics latch.
        sbit    cenclk,portph   ; Pulse CCTLCLK to load latch.
        rbit    cenclk,portph
        sbit    cdata,portbh    ; Load Centronics BUSY signal (high).
        ret

        .form   'UART and Input Prime Interrupt Handler'

        .ipt    6,uarint        ; UART Interrupt Vector
```

TL/DD/9977–44

```
                ;  This interrupt can indicate any of three conditions:
                ;    1)  A character has been sent, and the transmitter
                ;           is again ready (label "uarout").
                ;    2)  A character has been received (label "uartin").
                ;    3)  A Centronics INPUT PRIME event has been detected
                ;           (label "uarprm").


uarint:         push    psw
        push    A
        push    B
        push    K
        push    X

        ifbit   usel,ups        ; Check if UART selected.
        jmpl    uarchr          ; If so, go process a character interrupt.
        ifbit   enprm,cps       ; Check if PRIME interrupt enabled
        jmpl    uarprm          ;  from Centronics port.  If so,
                                ;  this means that the Centronics port
                                ;  is selected, and it must be a PRIME
                                ;  event.
        jsrl    hangup          ; Else, there is an error.  Stop.

uarchr:         ifbit   rbfl,enu        ; Check for Receiver interrupt.
        jmpl    uartin          ; Go process input character if so.
        ifbit   tbmt,enu        ; Check for Transmitter interrupt.
        jmpl    uarout          ; Go process output interrupt if so.
        jsrl    hangup          ; Else, there is an error.  Stop.


        .form   'UART Output Routine'

uarout:                         ; Here, the interrupt is because a character has just
                        ;   been sent and the transmitter buffer is now empty.
        ifbit   icpu,ups        ; Check if the CPU needs to be informed.
        jmpl    uicpu
        jmpl    unicpu
uicpu:  sbit    auack,alerth.b  ; Request main program to interrupt CPU for
                                ;  UART acknowledge.
        rbit    icpu,ups        ; Reset "Interrupt CPU" status on UART.
        jmpl    unicpu          ; Continue processing of interrupt.

unicpu:         ifbit   xonb,uflow      ; If XON mode selected,
        jsrl    setuar          ;  check UART handshake status and take any
                                ;  appropriate action.
        jmpl    uarret          ; Return.


        .form   'UART Input Routine'

uartin:                                 ; UART data input routine.

        ld      A,enur          ; Get image of error flags and RBIT9.
        ld      uinchr,rbuf     ; Get character.
        st      A,enrimg        ; Save image of ENUR for further processing.
                                ; Check for hardware-detected errors.
        and     A,#x'C0         ; Mask for error bits (Overrun/Framing).

        ld      X,uinchr        ; Prepare for parity check.
```

TL/DD/9977–45

```
        ld      B,#evntbl       ; Initialize B to point to Even Parity table.
        x       A,uframe        ; Parity processing depends on selected
                                ;  frame format, so branch to proper
        jid                     ;  parity processing routine.
        .pt     uiod8,uiev8,unopar,unopar
        .pt     uiod7,uiev7,uiod7,uiev7

                        ; Processing for 8-bit characters with parity.
uiod8:  ld      B,#oddtbl       ; For odd processing, change parity table base.
uiev8:  x       A,uframe        ; Recover cumulative errors in accumulator.
        ifbit   frm,A.b         ; Check for BREAK condition:  if framing error,
        jp      ufer8
        jp      u8nbrk
ufer8:  ifgt    uinchr,#0       ;  and data field is all zeroes,
        jp      u8nbrk
        ifbit   rbit9,enrimg    ;  and 9th bit also zero,
        jp      u8nbrk
        ifbit   onebrk,ups      ;  then check if this is the second
        jp      u82brk          ;   consecutive BREAK.
        sbit    onebrk,ups      ;  If not, then flag only the framing error,
        jp      u8dopr          ;   and do not report break status yet.
u82brk:         sbit    brk,A.b         ;  If so, then set Break bit in error image and
        rbit    eri,enui        ;   disable UART receiver until re-selected.
        sbit    brkmd,ups       ;   Also show receiver disabled in UPS byte.
u8nbrk:         rbit    onebrk,ups
u8dopr:         ifbit   X,[B].b         ; Check parity of 8-bit character.  Set "par"
        sbit    par,A.b         ;  bit of Accumulator if it would be incorrect
                                ;  without parity bit.
        ifbit   rbit9,enrimg    ; Check parity bit for 8-bit character.  Toggle
        xor     A,#x'20         ;  parity error indication if set.
uinpok:         ifeq    A,#x'00         ; Branch based on presence of error.
        jmpl    uingd
        jmpl    uinerc

                        ; Processing for 7-bit characters with parity.
uiod7:  ld      B,#oddtbl       ; For odd processing, change parity table base.
uiev7:  x       A,uframe        ; Recover cumulative errors in accumulator.
        ifbit   frm,A.b         ; Check for BREAK condition:  if framing error,
        jp      ufer7
        jp      u7nbrk
ufer7:  ifgt    uinchr,#0       ;  and data field is all zeroes (incl. parity),
        jp      u7nbrk
        ifbit   onebrk,ups
        jp      u72brk
        sbit    onebrk,ups
        jp      u7dopr
u72brk:         sbit    brk,A.b         ;  then set Break bit in error image and
        rbit    eri,enui        ;  disable receiver.
        sbit    brkmd,ups       ;  Also show receiver disabled in UPS byte.
u7nbrk:         rbit    onebrk,ups
u7dopr:         rbit    7,uinchr        ; Seven-bit data:  clear parity bit in memory.
        ifbit   X,[B].b         ; Perform bit-table lookup:  1 means error.
        jp      uipe7
        jmpl    uinpok
uipe7:  sbit    par,A.b         ; Set parity error indication in A.
        jmpl    uinerc

                        ; For 8-bit character frames with no parity:
unopar:         x       A,uframe        ; Restore frame value to UFRAME, and continue
                                ;  (no parity check in these modes).
```

TL/DD/9977–46

```
        ifbit   frm,A.b         ; Check for BREAK condition:  if framing error,
        jp      uferr
        jp      unbrk
uferr:  ifgt    uinchr,#0       ;  and data field is all zeroes (incl. parity),
        jmp     unbrk
        ifbit   onebrk,ups      ;  then BREAK condition:  if previous character
        jmp     un2brk
        sbit    onebrk,ups      ;    was not a BREAK, then just note this one.
        jp      unobrk
un2brk:         sbit    brk,A.b         ;  If it was, then set Break bit in error image
        rbit    eri,enui        ;  and disable receiver.
        sbit    brkmd,ups       ;  Also show receiver disabled in UPS byte.
unbrk:  rbit    onebrk,ups
unobrk:         jmpl    uinpok

uingd:                  ; Here, a good character was received.  Start buffer
                        ;  processing.
        ld      A,uinchr        ; Get character again.
        ld      K,#topad        ; Reg. K gets buffer top address.

                        ; Test whether there is room for another byte
                        ;  in the data buffer.
        ifbit   full,bstat      ; If FULL bit set,
        jmpl    uinerf          ;  process this character as an error
                                ;  (Buffer Overflow).

        ld      B,cadin         ; Get current buffer input address.
        xs      A,[B+].b        ; Store character in table.
        jp      uin0            ; If skip,
        ld      B,#botad        ;   then wrap input pointer to beginning
uin0:   ld      cadin,bl.b      ;   of buffer;  else just increment it.

uinl:   inc     numchr          ; Increment number of characters.
        ifgt    pascnt,numchr   ; Check if buffer full enough to send.
        jmpl    uinex           ;     No:  end of service.

        sbit    pass,bstat      ;     Yes: indicate buffer ready to pass.
        sbit    4,lcvs          ; (DEBUG: report status in LCD Contrast latch.)
        ld      portah,lcvs
        sbit    lcvclk,portbh
        rbit    lcvclk,portbh

        ifgt    stpcnt,numchr   ; Check if buffer too full for more
                                ;     host characters.
        jmpl    uinex           ;  No:  end of service.

        sbit    cus,ups         ;  Yes:  set UART input port status busy.
        sbit    stop,bstat      ;        set Buffer Status as "STOPPED".
        jsrl    dtroff          ;        set DTR handshake appropriately.
        ifbit   eti,enui        ;        check if UART transmitter busy.
        jp      uin2
        ifbit   xonb,uflow      ;          if not, then if XON mode selected,
        jsrl    setuar          ;          then invoke flow control routine.
                                ;          (otherwise it will happen on next
                                ;           UART transmitter interrupt
                                ;           automatically).
uin2:
        sbit    5,lcvs          ; (DEBUG: report status in LCD Contrast latch.)
        ld      portah,lcvs
        sbit    lcvclk,portbh
```

TL/DD/9977–47

```
        rbit    lcvclk,portbh

        ifeq    numchr,#bufsiz  ; Check if buffer completely full.
        sbit    full,bstat      ; Yes:  set condition.

        jmpl    uinex

uinerc:                         ; Character error handler.
        ifbit   aerr,alerth.b   ; If an error has already been posted,
        jp      uinmce          ;   handle as a multiple error.
        jmpl    uinlce          ; Else, report single error.

uinmce:         sbit    errovf,errfgs   ; Update error conditions byte to also report
                                ;  a lost error.
        or      errfgs,A.b      ; OR in the errors from this character.
        sbit    cus,ups         ;  Yes:  set UART input port status busy.
        ifbit   eti,enui        ;        check if UART transmitter busy.
        jp      uinmc2
        ifbit   xonb,uflow      ;          if not, then if XON mode selected,
        jsrl    setuar          ;          then invoke flow control routine.
                                ;          (otherwise it will happen on next
                                ;           UART transmitter interrupt
                                ;           automatically).
uinmc2:         jsrl    dtroff          ; Remove DTR handshake if flow mode requires it.
        rbit    eri,enui        ; Disable UART input interrupt until
                                ;  re-initialized by CPU.
        sbit    mcemd,ups       ;  Also flag receiver disabled in UPS byte.
        jmpl    uinex           ; Return from the interrupt.

uinlce:
        sbit    aerr,alerth.b   ; Request CPU interrupt from main program.
        st      A,errfgs        ; Report error flags from Accumulator.
        ld      errchr,uinchr   ; Report error character.
        ld      fshlim,numchr   ; Establish limit on future flushes.
        jmpl    uinex           ; Return from the interrupt.

uinerf:                         ; FULL error handler:  invoked if HPC's data buffer
                        ;  overflows.

        sbit    7,lcvs          ; (DEBUG:  report error in LCD Contrast latch.)
        ld      portah,lcvs
        sbit    lcvclk,portbh
        rbit    lcvclk,portbh

        ifbit   aerr,alerth.b   ; If an error has already been posted,
        jp      uinmef          ;   handle as a multiple error.
        jmpl    uinlef          ; Else, report single error.

uinmef:         sbit    bufovf,errfgs   ; Signal buffer overflow as another error.
        sbit    errovf,errfgs   ; Update error conditions byte to also report
                                ;  a lost error.
        sbit    cus,ups         ;  Set UART input port status busy.
        rbit    luss,ups        ;  (This is done to force flow control action.)
        ifbit   eti,enui        ;  Check if UART transmitter busy.
        jp      uinme2
        ifbit   xonb,uflow      ;  If not, then if XON mode selected,
        jsrl    setuar          ;   then invoke flow control routine.
                                ;   (otherwise it will happen on next
                                ;    UART transmitter interrupt automatically).
uinme2:         jsrl    dtroff          ; Remove DTR handshake if flow mode needs it.
```

TL/DD/9977–48

```
        rbit    eri,enui        ; Disable UART input interrupt until
                                ;  re-initialized by CPU.
        sbit    mcemd,ups       ;  Also flag receiver disabled in UPS byte.
        jmpl    uinex           ; Return from the interrupt.

uinlef:         sbit    aerr,alerth.b   ; Signal an error.
        ld      errfgs,#x'10    ; Report buffer overflow as reason.
        ld      errchr,uinchr   ; Place character in ERRCHR slot for report to
                                ;   CPU.
        ld      fshlim,numchr   ; Establish limit on future flushes.
        sbit    cus,ups         ;  Set UART input port status busy.
        rbit    luss,ups        ;  (This is done to force flow control action.)
        ifbit   eti,enui        ;  Check if UART transmitter busy.
        jp      uinlf2
        ifbit   xonb,uflow      ;  If not, then if XON mode selected,
        jsrl    setuar          ;   then invoke flow control routine.
                                ;   (otherwise it will happen on next
                                ;    UART transmitter interrupt automatically).
uinlf2:         jsrl    dtroff          ; Remove DTR handshake if flow mode needs it.
        jmpl    uinex           ; Return from the interrupt.


uinex:                  ; Exit from UART input character processing.
        jmpl    uarret          ; Return.

        ; Parity Bit Lookup Table

evntbl:         .byte  X'96,X'69,X'69,X'96,X'69,X'96,X'96,X'69
        .byte  X'69,X'96,X'96,X'69,X'96,X'69,X'69,X'96
oddtbl:         .byte  X'69,X'96,X'96,X'69,X'96,X'69,X'69,X'96
        .byte  X'96,X'69,X'69,X'96,X'69,X'96,X'96,X'69
        .byte  X'96,X'69,X'69,X'96,X'69,X'96,X'96,X'69
        .byte  X'69,X'96,X'96,X'69,X'96,X'69,X'69,X'96
;
; A one in the table means incorrect parity for the mode,
;  the mode being expressed as the base address (evntbl or oddtbl).
        .form   'Centronics INPUT PRIME'


                        ; Centronics INPUT PRIME service.
uarprm:         sbit    aprime,alert.b  ; Set PRIME bit in Alert mailbox to Main prog.
        sbit    cbusy,cps       ; Set BUSY bit in Centronics status byte.
        jsrl    setcen          ; Go set up Centronics port itself.
        rbit    uart,enir       ; Disable interrupt until it goes away.
        jmpl    uarret          ; Return.


uarret:         pop     X       ; Common return from UART interrupt.
        pop     K
        pop     B
        pop     A
        pop     psw
        reti

        .form   'Subroutine to Wait for OBUF Empty'

;       RDWAIT subroutine:  waits until the CPU has read a byte from the
;                           UPI interface.

rdwait:         ifbit   rdrdy,upic      ; Check to see if OBUF register is full.
```

TL/DD/9977–49

```
        ret
        jp      rdwait

        .form   'Write to Panel Subroutine'

                        ; Write Panel subroutine.
                        ; Used only at initialization or to report a
                        ; fatal protocol error, since it performs
                        ; the timing delay using timer T6 without interrupts.
                        ;   (Panel RS signal must be set up previously in the
                        ;    LCV latch by the calling routine.)

wrpnl:  comp    A               ; Complement value for bus.
        st      A,portah        ; Put value on panel bus.
        rbit    pnlclk,portbl   ; Set Panel Clock low,
        sbit    pnlclk,portbl   ;  then high again;
                                ;  pulse width approx.
                                ;  1.2 microsec.

                        ; Wait for another
                        ;  4.9 milliseconds (twice).
        ld      t6,#13000       ; Twice 4.9 milliseconds.
        rbit    t6stp,pwmdh     ; Start timer T6.
wrplp:  ifbit   t6pnd,pwmdh     ; Wait for PND to be set.
        jp      wrpgo
        jp      wrplp
wrpgo:  sbit    t6stp,pwmdh     ; Stop timer T6.
        sbit    t6ack,pwmdh     ; Clear T6 PND bit.
        ret                     ; Return from subroutine.

        .form   'Set up UART flow control/output'

setuar:                         ; Subroutine SETUAR:  checks status of UART output
                        ;   section, and initiates a transfer if needed.
        ld      A,ups           ; Check if UART handshake status needs update.
        and     A,#x'03
        shl     A
        .odd
        jidw
        .ptw    usmat,usnmat,usnmat,usmat

                        ; Here, UART status last sent does not match
                        ;  current status.  Needs flow control action.
usnmat:
        ifbit   cus,ups
        jmpl    ustop
ugo:    ld      X,#xon          ; Get XON (Control-Q) code.
        jsrl    uecsnd          ; Format it and send.
        rbit    luss,ups
        jmpl    sturet          ; Return.
ustop:  ld      X,#xoff         ; Get XOFF (Control-S) code.
        jsrl    uecsnd          ; Format it and send.
        sbit    luss,ups
        jmpl    sturet          ; Return.

usmat:                  ; No flow control needed.  Check if CPU character is
                        ;  waiting to be sent.
        ifbit   schr,ups
        jmpl    uscpc
```

TL/DD/9977–50

```
unopnd:                         ; Here, no characters pending to be sent.  Turn off
                        ;  transmitter interrupt and return.
        rbit    eti,enui        ; Turn off transmitter interrupts.
        jmpl    sturet          ; Return.

uscpc:                  ; Here, a character is waiting to be sent from CPU.
        ld      X,uschr         ; Get character.
        jsrl    uecsnd          ; Format character for current frame and send.
        rbit    schr,ups        ; Remove character send request.
        sbit    icpu,ups        ; Set CPU interrupt request on completion.
        jmpl    sturet          ; Return.

sturet:         ret                     ; Return from subroutine.

        .form   'Format and transmit UART character'

uecsnd:                         ; Subroutine to encode a character according to the
                        ;   currently-selected frame format and send it.
                        ;   Character is passed in Register X.
        ld      B,#evntbl
        rbit    xbit9,enu
        ld      A,uframe        ; Jump based on frame format.
        jid
        .pt     su8odd,su8evn,su8,su8
        .pt     su7odd,su7evn,su7odd,su7evn

su8odd:         ld      B,#oddtbl
su8evn:         ifbit   X,[B].b
        sbit    xbit9,enu
        ld      tbuf,X.b
        sbit    eti,enui
        ret

su7odd:         ld      B,#oddtbl
su7evn:         ifbit   X,[B].b
        xor     X.b,#x'80       ; Toggle parity to ignore bad top bit.
        ld      tbuf,X.b
        sbit    eti,enui
        ret

su8:    ld      tbuf,X.b
        sbit    eti,enui
        ret

        .form   'DTR Handshake Routines'

                        ; Subroutine DTROFF - Sets printer not ready using DTR.
dtroff:         ifbit   dtrbl,uflow     ; Action taken depends on UFLOW mode.
        jp      doff            ; If DTR is in a permanent state, return.
        ret
doff:   ifbit   dtrb0,uflow
        jp      d2off
        sbit    dtr,portbl      ; For low-active DTR mode.
        ret
d2off:  rbit    dtr,portbl      ; For high-active DTR mode.
        ret

                        ; Subroutine DTRON - Sets printer ready using DTR.
dtron:  ifbit   dtrbl,uflow     ; Action taken depends on UFLOW mode.
        jp      dton            ; If DTR is in a permanent state, return.
```

TL/DD/9977–51

```
        ret
dton:   ifbit   dtrb0,uflow
        jp      d2on
        rbit    dtr,portbl      ; For low-active DTR mode.
        ret
d2on:   sbit    dtr,portbl      ; For high-active DTR mode.
        ret

        .end    start
```

TL/DD/9977–52

Lit. # 100551

LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

National Semiconductor Corporation
1111 West Bardin Road
Arlington, TX 76017
Tel: 1(800) 272-9959
Fax: 1(800) 737-7018

National Semiconductor Europe
Fax: (+49) 0-180-530 85 86
Email: cnjwge@tevm2.nsc.com
Deutsch Tel: (+49) 0-180-530 85 85
English Tel: (+49) 0-180-532 78 32
Français Tel: (+49) 0-180-532 93 58
Italiano Tel: (+49) 0-180-534 16 80

National Semiconductor Hong Kong Ltd.
13th Floor, Straight Block,
Ocean Centre, 5 Canton Rd.
Tsimshatsui, Kowloon
Hong Kong
Tel: (852) 2737-1600
Fax: (852) 2736-9960

National Semiconductor Japan Ltd.
Tel: 81-043-299-2309
Fax: 81-043-299-2408