A Software Driver for the HPC Universal Peripheral Interface Port

National Semiconductor
Application Note 550
Brian Marley
April 1992

ABSTRACT

This application note covers the use of the National Semiconductor HPC46083 High-Performance microController as an intelligent Peripheral Interface and Interrupt controller for another "Host" CPU, using its 8-bit or 16-bit parallel UPI (Universal Peripheral Interface) Port. Included in the discussion is the source text of an HPC driver program, which can be tailored as an "executive" for a wide variety of HPC tasks. A simple application is built from this software, which interfaces a National NS32CG16 CPU to a typical front panel (LED indicators, LCD alphanumeric display, pushbuttons and beeper).

1.0 INTRODUCTION

The National Semiconductor HPC family of microcontrollers includes as a feature the ability to be slaved to another "Host" processor over that processor's memory bus. This feature, called the Universal Peripheral Interface (or UPI) Port, allows:

1. Transfer of either 8-bit or 16-bit data in a single bus transaction,

2. Polling to determine the status of the port from either side (Ready for Write/Ready for Read), and

3. Interruption of the host by the HPC with full vectoring.

The HPC, then, can serve as a front-end controller for the host, freeing it from control and/or communication tasks that might burden its capacity for interrupt service, and providing vectored interrupting for higher-level (and therefore less frequent) communication.

2.0 THE UPI PORT

2.1 Internal Structure

Figure 1 shows the internal structure of the UPI Port. It connects via three registers to the HPC's on-chip data bus, and via a set of pins (Port A) to the host's bus. The control interface between the HPC and the host consists of two low-active strobe signals ($\overline{URD}$ and $\overline{UWR}$) and an address signal (UA0) output by the host, and two handshake signals ($\overline{RDRDY}$ and $\overline{WRRDY}$) output from the HPC.
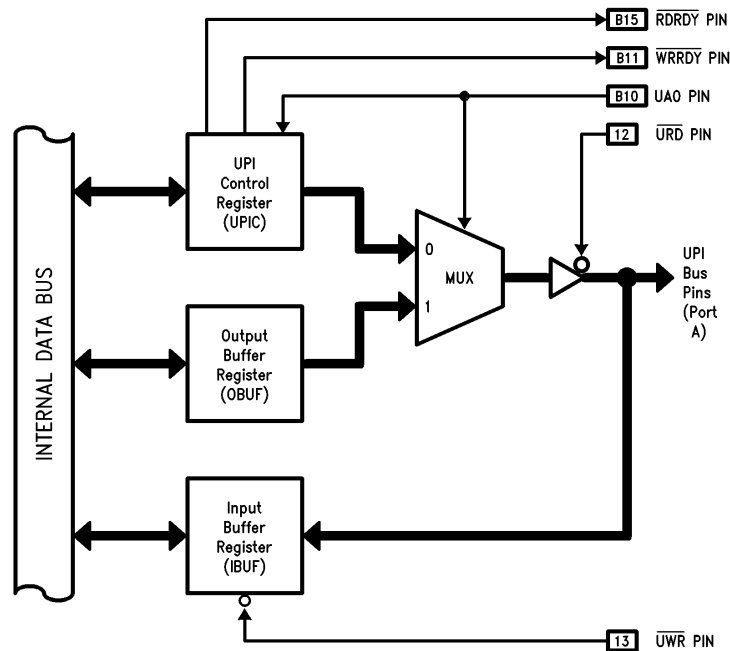

FIGURE 1. UPI Internal Structure

TL/DD/9976–1

The UPI Port may be configured either as a 16-bit bus (using all of Port A: pins A0–A15) or as an 8-bit bus (pins A0–A7), allowing pins A8–A15 to be used as general-purpose bit-programmable I/O pins. This selection is made by HPC firmware.

2.2 Basic Operations

Three types of operation may be performed over the UPI Port:

1. Transfer of a byte or word of data from the host to the HPC's IBUF register. This is called a "UPI Write" operation.

2. Transfer of a byte or word of data from the HPC's OBUF register to the host. This is called a "UPI Read" operation.

3. Polling by the host to determine whether the HPC is ready for the next UPI Write or UPI Read operation. This involves the host reading the UPIC (UPI Control) register, which contains the states of the $\overline{WRRDY}$ and $\overline{RDRDY}$ pins as two of its bits.

As shown in *Figure 2*, whenever the host writes to the HPC (by pulsing the $\overline{UWR}$ signal low) data is latched into the HPC's IBUF register. At this time also, the value on the UA0 pin is latched into the UPIC (UPI Control) register, allowing HPC firmware to route the data just written. (For example, this bit can be used by the HPC firmware to distinguish between commands and data written to it.) The rising edge of $\overline{UWR}$ is detected by an edge-trigger circuit on-chip, which may be used to trigger an interrupt or for polling, to alert the HPC firmware to the presence of new data. The $\overline{WRRDY}$ handshake signal, normally low, goes high until the HPC firmware has sampled the data written to it (by reading internally from the IBUF register).

Figure 3 shows the sequence of events in reading data from the HPC. The transfer starts when the HPC writes a value to the internal OBUF register. The $\overline{RDRDY}$ handshake signal, normally high, goes low to indicate that data is present for the host. (This pin can be used to interrupt the host as well.) By pulsing the $\overline{URD}$ pin low while holding the UA0 pin to a "1", the host reads the contents of the OBUF register, and the $\overline{RDRDY}$ pin goes back high.

The polling operation *(Figure 4)* allows the host to monitor the $\overline{RDRDY}$ and $\overline{WRRDY}$ conditions as data bits, by pulsing the $\overline{URD}$ pin low with a "0" held on the UA0 pin. This effectively reads from the UPIC register; the $\overline{WRRDY}$ condition appears on bit 0 (the least-significant bit), and the $\overline{RDRDY}$ condition appears on bit 1 (the next most significant bit). Polling in this manner does not affect the state of the $\overline{RDRDY}$ bit.
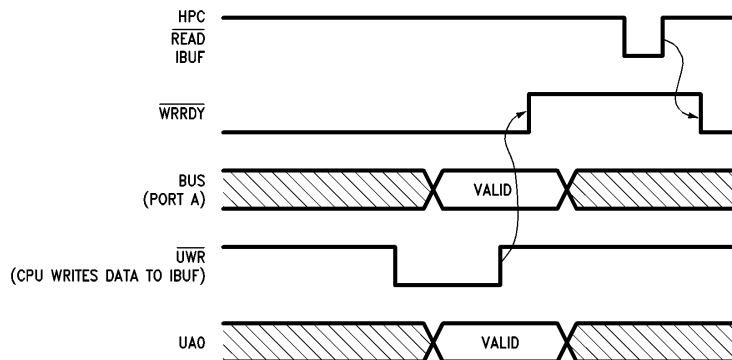
TL/DD/9976–2

FIGURE 2. UPI Write Operation

HPC
$\overline{\text{WRITE}}$
(TO OBUF)

$\overline{\text{RDRDY}}$

BUS
(PORT A)   VALID

$\overline{\text{URD}}$
(CPU READ FROM OBUF)

$\overline{\text{UA0}}$ VALID (HIGH)
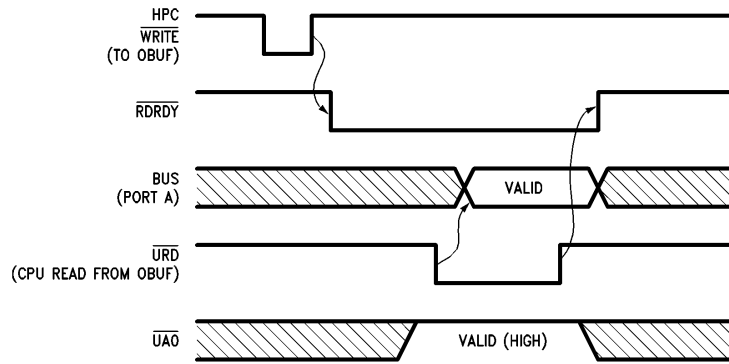
TL/DD/9976–3

FIGURE 3. UPI Read Data Operation

$\overline{\text{RDRDY}}$ (NOT AFFECTED)

BUS
(PORT A)   VALID

$\overline{\text{URD}}$
(CPU READ FROM UPIC)

$\overline{\text{UA0}}$ VALID (LOW)
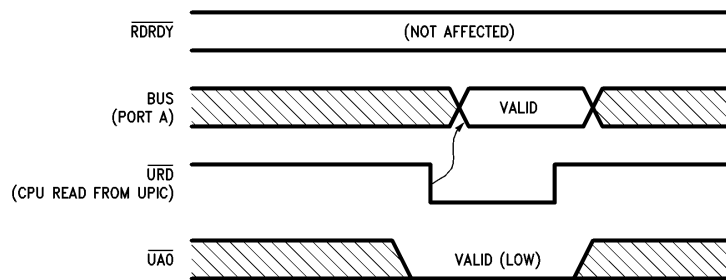
TL/DD/9976–4

FIGURE 4. UPI Poll Operation

2.3 Typical Hardware Configurations

Typical connections between the host and the HPC are shown in *Figures 5* through *7*.

2.3.1 Polled Synchronization

In the simplest case *(Figure 5)*, the $\overline{\text{WRRDY}}$ and $\overline{\text{RDRDY}}$ signals are not used, and the host synchronizes itself with the HPC strictly by polling the UPIC register for the Read Ready and Write Ready conditions. The only additional logic always required is a pair of OR gates to activate $\overline{\text{URD}}$ and $\overline{\text{UWR}}$ only when the HPC is selected by the host's address decoder. Depending on the host, it may also be necessary to add WAIT states, as is often required in peripheral interfaces to match the bus timing characteristics of the two ends.

Sophisticated synchronization schemes are not available using this simple an interface, but it does save the HPC $\overline{\text{RDRDY}}$ and $\overline{\text{WRRDY}}$ pins for any other general-purpose I/O functions.

2.3.2 Interrupt-Driven Synchronization

Assuming that the host has interrupt control capability, the circuit above can be enhanced to implement an interrupt-driven synchronization scheme, as shown in *Figure 6*. A falling edge on either $\overline{\text{RDRDY}}$ or $\overline{\text{WRRDY}}$ will trigger an interrupt to the host, informing it when the HPC becomes ready for either direction of data transfer. No additional logic is required (except for possible buffering or inversion), but only dedication of the $\overline{\text{WRRDY}}$ and/or $\overline{\text{RDRDY}}$ pins for the interrupt function. It is not necessary for both RDRDY and WRRDY conditions to trigger interrupts; one can be polled and the other interrupt-driven, as dictated by the require-ments of the system and the structure of the host and HPC software. Also, depending on the host, it is often possible for the HPC itself to provide interrupt vectoring, thus eliminating the need for an external interrupt controller entirely. The approach taken in the driver program, described below, implements the HPC as the interrupt controller, with interrupts asserted only by the $\overline{\text{RDRDY}}$ pin.

2.3.3 Hardware Synchronization

Figure 7 shows the connections required to implement hardware synchronization between the host and the HPC. In this scheme, there is no host software involved in synchronizing with the HPC; if the host attempts a UPI transfer for which the HPC is not prepared, the host is held in "Wait states" until the HPC is ready. Note that the UPIC register is an exception; Wait states are not to be inserted when the CPU polls the UPI port's status (UA0 = 0).

The main advantage of this scheme is speed: the CPU and HPC transfer data as fast as they can both run the transfer loop. (One will generally find that the HPC stays ahead of the CPU; the CPU tends to be in the critical path due to more complex buffer management algorithms.) The main disadvantage is that if the HPC is allowed to be interrupted in the middle of the transfer, the CPU is not free to do anything else at all, including servicing its own interrupts.

In addition to the logic to detect when to hold the host (at the bottom of the figure), additional gating is required on the $\overline{\text{UWR}}$ signal, to prevent it from being asserted until the $\overline{\text{WRRDY}}$ signal is active. This is required because the IBUF register of the HPC is a fall-through latch, and its contents would be lost if $\overline{\text{UWR}}$ were allowed to go active too soon.
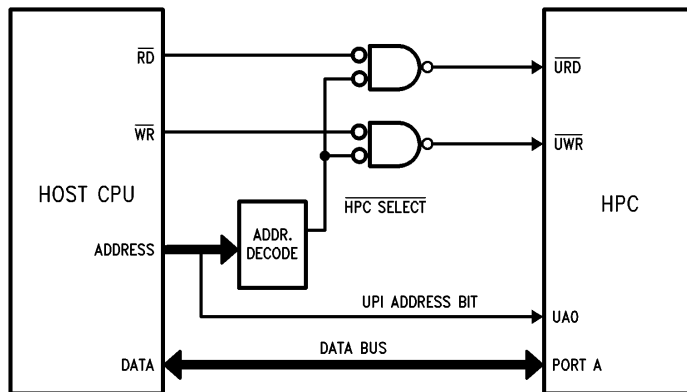
TL/DD/9976-5

FIGURE 5. Polling Interface

FIGURE 6. Interrupt-Driven Interface

TL/DD/9976–6

FIGURE 7. Hardware-Synchronized Interface
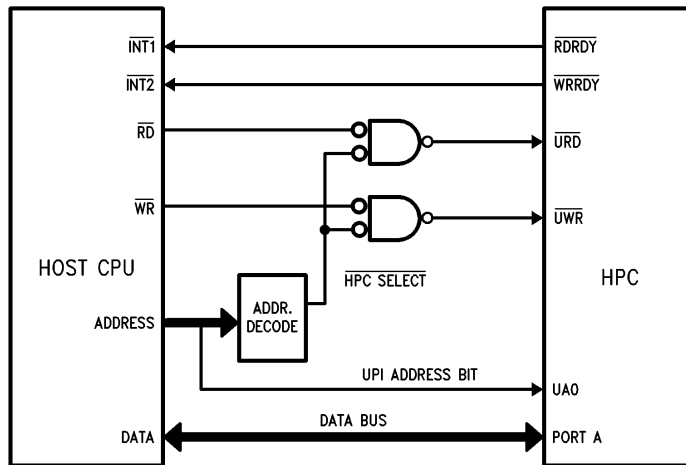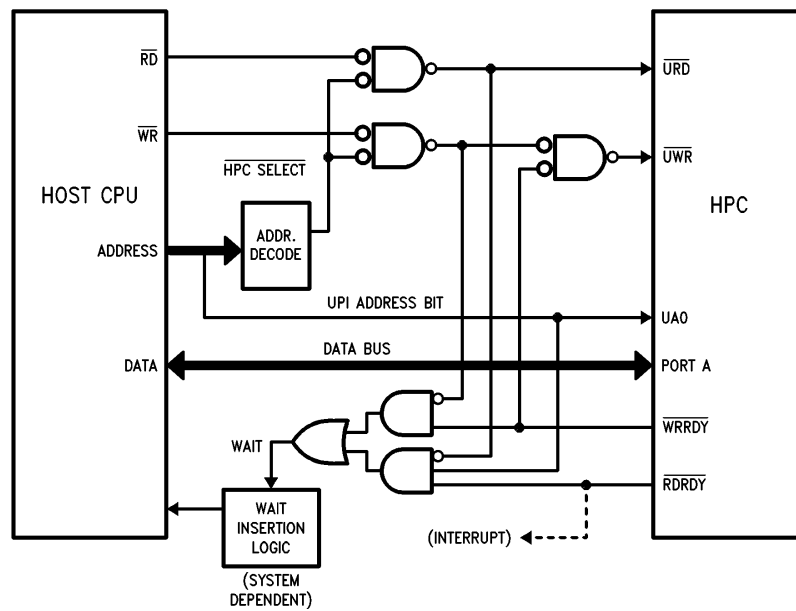
TL/DD/9976–7

Figures 8 and 9 illustrate the timing involved in hardware synchronization. Figure 8 shows the host attempting two UPI Read accesses in quick succession; the second Read access is held pending until the HPC has supplied the data. Figure 9 shows the host attempting two UPI Write accesses in quick succession; it is held in Wait states (with the $\overline{UWR}$ signal suppressed) until the HPC has emptied the first value from the IBUF register.

This scheme and the interrupt-driven scheme above are not mutually exclusive; as shown in Figure 6, one might tie $\overline{RDRDY}$ or $\overline{WRRDY}$, or both, to CPU interrupts. The application hardware described implements both schemes, leaving CPU software the option of using hardware synchronization or not. The driver program in the HPC operates the same, independent of the option used.
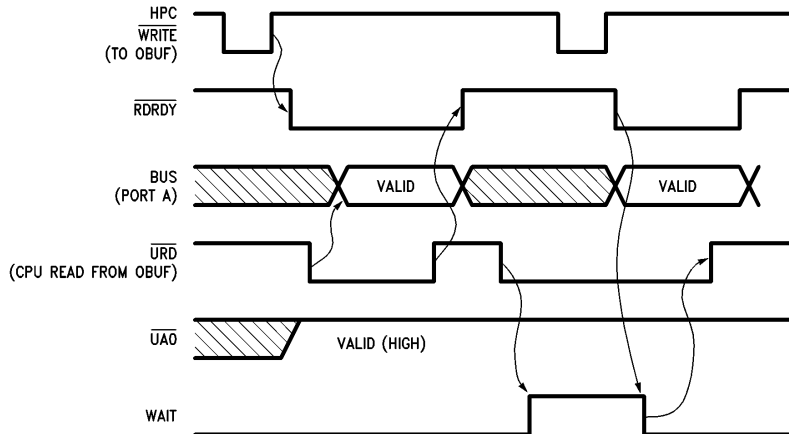
TL/DD/9976–8

FIGURE 8. Hardware Synchronization: Read Operations

TL/DD/9976–9
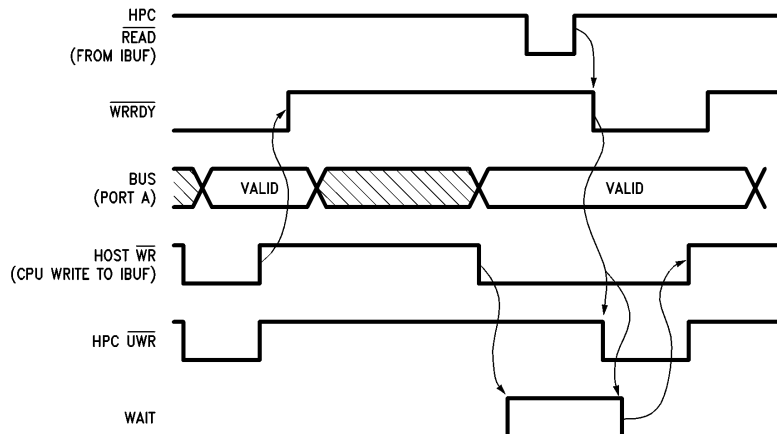
FIGURE 9. Hardware Synchronization: Write Operations

6

3.0 A UPI DRIVER AND SAMPLE APPLICATION

The circuit and program described below implement an interface between the HPC and a National microprocessor, the NS32CG16, as the host CPU. The UPI port is configured to be 8 bits wide. The hardware supports both interrupt-driven ($\overline{RDRDY}$ only) and hardware synchronization, as well as polling.

In order to demonstrate some real commands to support, a set of simple interfaces is attached to the HPC, typical of a front panel.

— Up to 8 pushbuttons
— Up to 8 LED indicators
— A 16-character alphanumeric LCD display
— A speaker for ''beeps'' on alert conditions or input errors
— A real-time clock interrupt function, giving the CPU the means to measure time intervals accurately.

This application by itself is admittedly not enough to justify the presence of an HPC in a system, but it is a simple application, and we expect that this will often be part of the HPC's job. For a much more comprehensive application, which includes this one as a subset, see the next application note in this series: ''The HPC as a Front-End Processor''.

We will describe in this section a specific set of hardware and software, and a UPI command and response protocol to make these interfaces play.

3.1 UPI Port Connections to NS32CG16

The attached schematic shows the HPC UPI port as it has been used a real application. On Sheet 1, a block diagram is given, showing the components involved. The CPU is an NS32CG16 microprocessor, running at a 15 MHz clock rate (crystal frequency 30 MHz). The HPC component is the HPC46083, running at a crystal frequency of 19.6608 MHz.

It would be unrealistic to present only the UPI interface section, since tradeoffs and implementation considerations abound when dealing with fast processors and large addressing spaces. For this reason, we include on sheets 5, 6 and 7 the circuitry involved in NS32CG16 address decoding and dynamic RAM control.

The $\overline{UREAD}$ and $\overline{UWRITE}$ UPI strobes are generated for the HPC in area B1 of Sheet 6. In addition, the latched CPU address bit BA09 is used as the UA0 addressing bit.

Hardware and Interrupt synchronization are accomplished as follows. On Sheet 6, area D8, the HPC signals $\overline{URDRDY}$ and $\overline{UWRRDY}$ enter a synchronizer, and emerge as $\overline{URDRDYS}$ and $\overline{UWRRDYS}$. The $\overline{URDRDYS}$ signal goes to the CPU as its Maskable Interrupt signal (Sheet 5, area C8). After gating, which yields $\overline{URDRDYSQ}$ and $\overline{UWRRDYSQ}$, they enter the PAL16L8 in area C7 of Sheet 6. This PAL's relevant outputs are $\overline{WAIT1}$ and $\overline{WAIT2}$, which go to the CPU for Wait State generation, and $\overline{ACWAIT}$, which also goes to the CPU (as $\overline{CWAIT}$) after passing through the PAL20R8 device in area D4 of Sheet 6.

In addition, the HPC provides from Timer T4 a square wave at approximately 68 kHz, which triggers refreshes of dynamic RAM. The signal involved is called ''68 kHz'', and goes from the HPC on Sheet 4, area D1, to Sheet 6, area D8. Note that the detector in area D7 is held on at Reset, to preserve RAM contents by continuous refreshing while the HPC is being reset.

3.1.1 Schematic

UPI Demo Functional Block Diagram

TL/DD/9976–10

HPC Microcontroller

TL/DD/9976-11

8

HPC I/O

PORTD0–PORTD7

TO LCD CONTROLLER
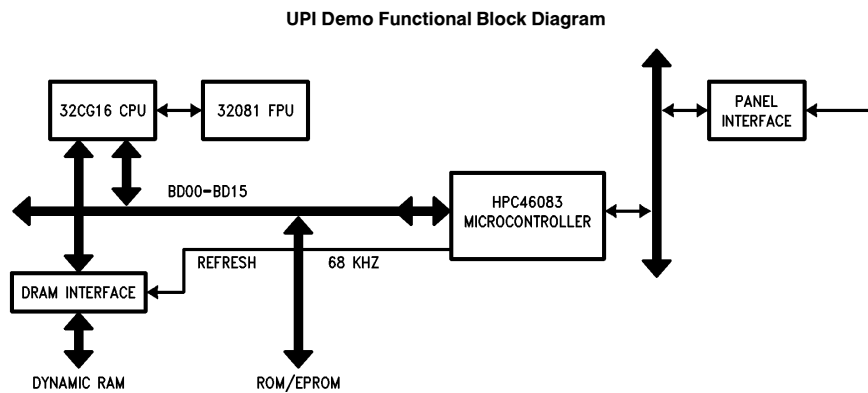
TL/DD/9976–12
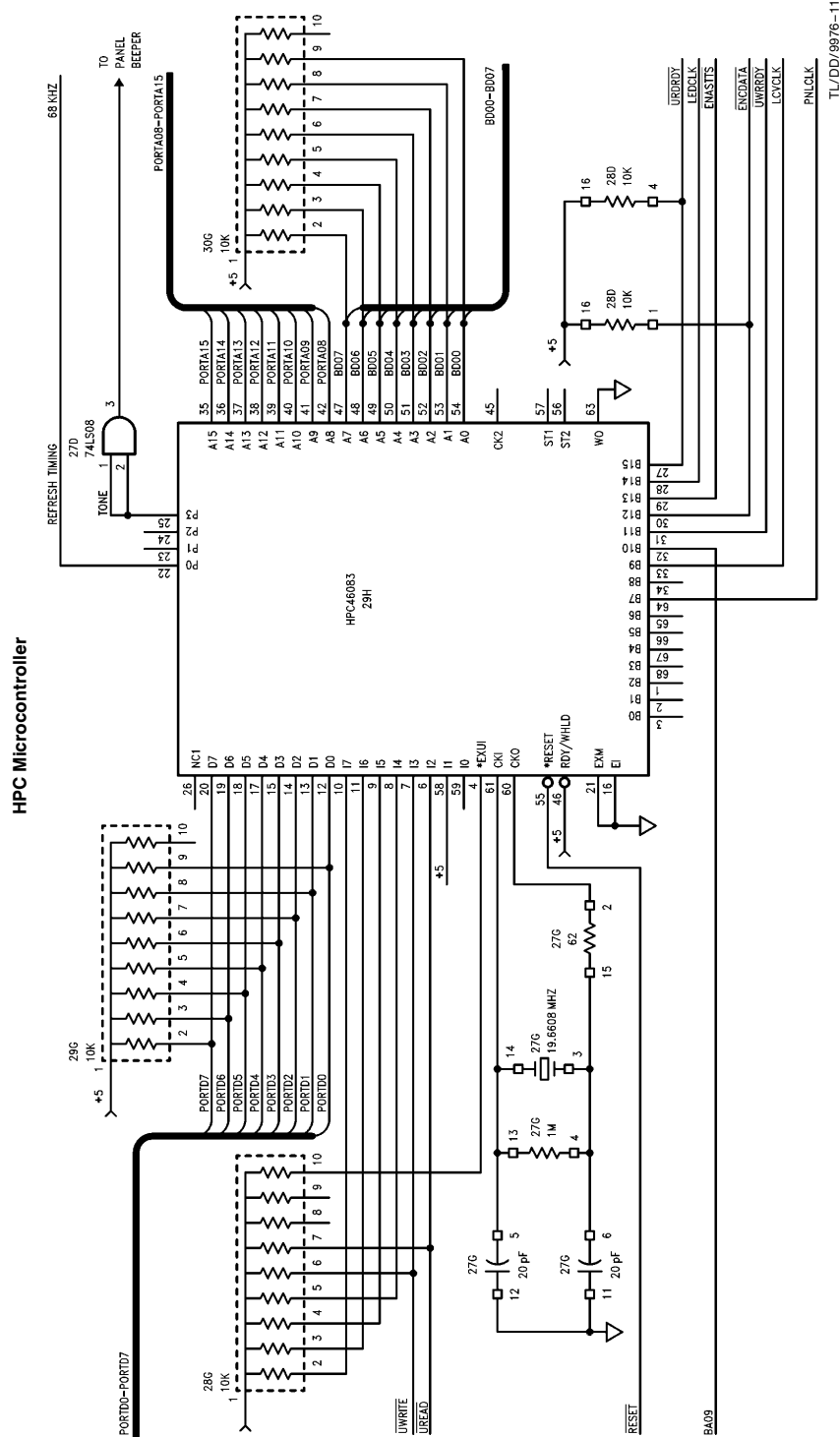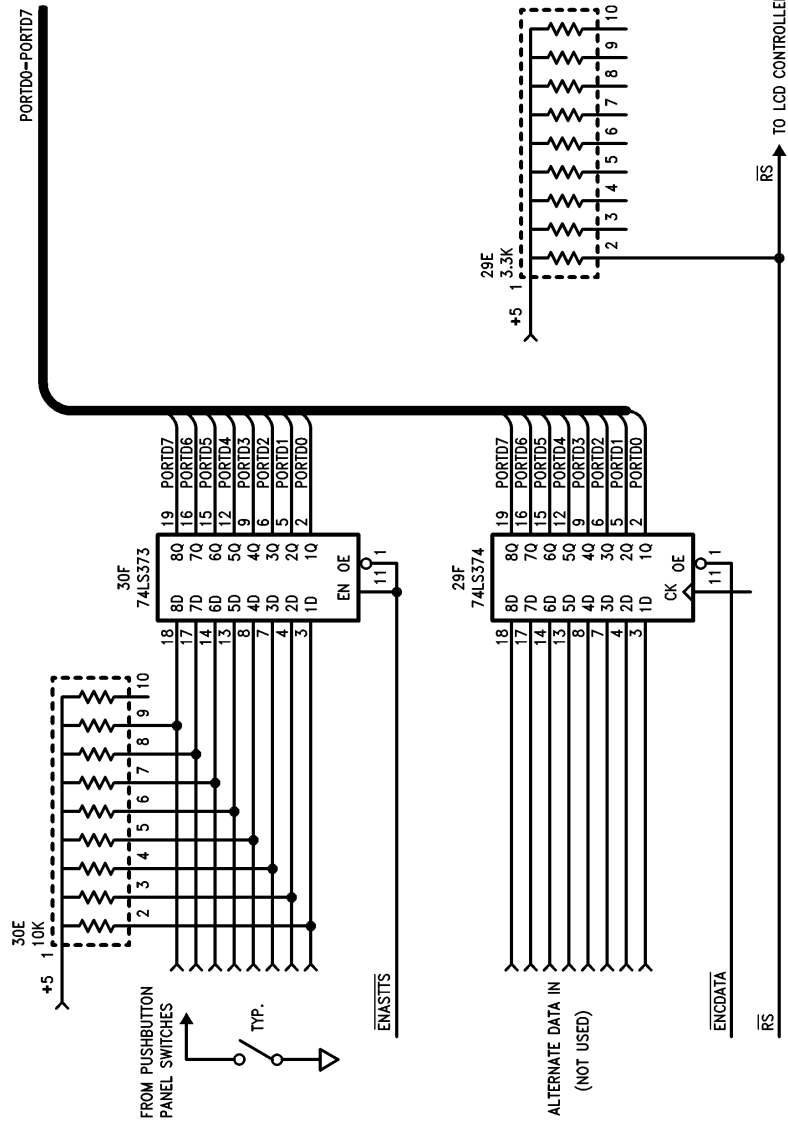
Panel I/O Interface

TL/DD/9976–13

TYPICAL LCD CONNECTIONS

HD44780 LCD CONTROLLER

SIMPLE D/A
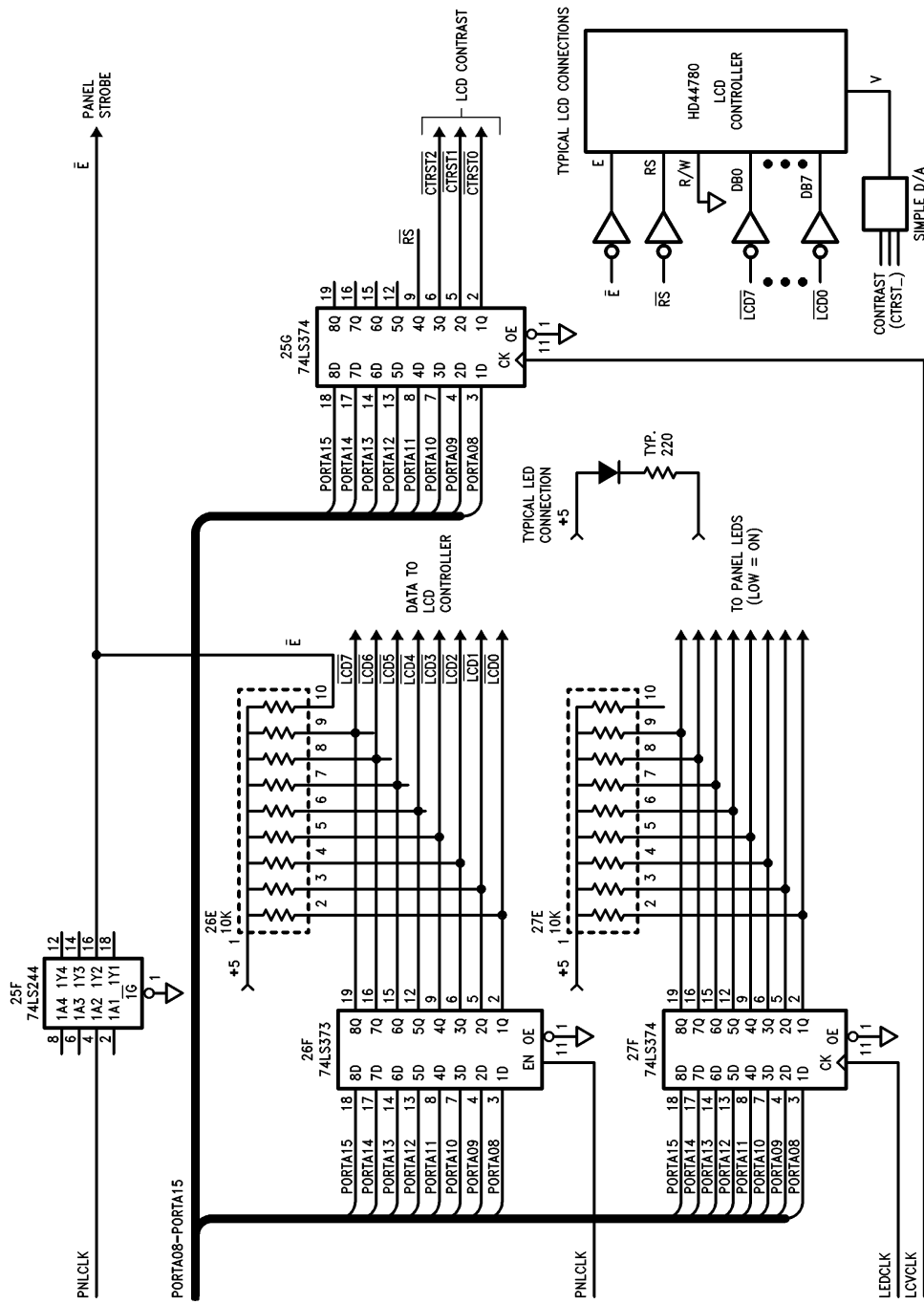
CPU Cluster and Buffering

11
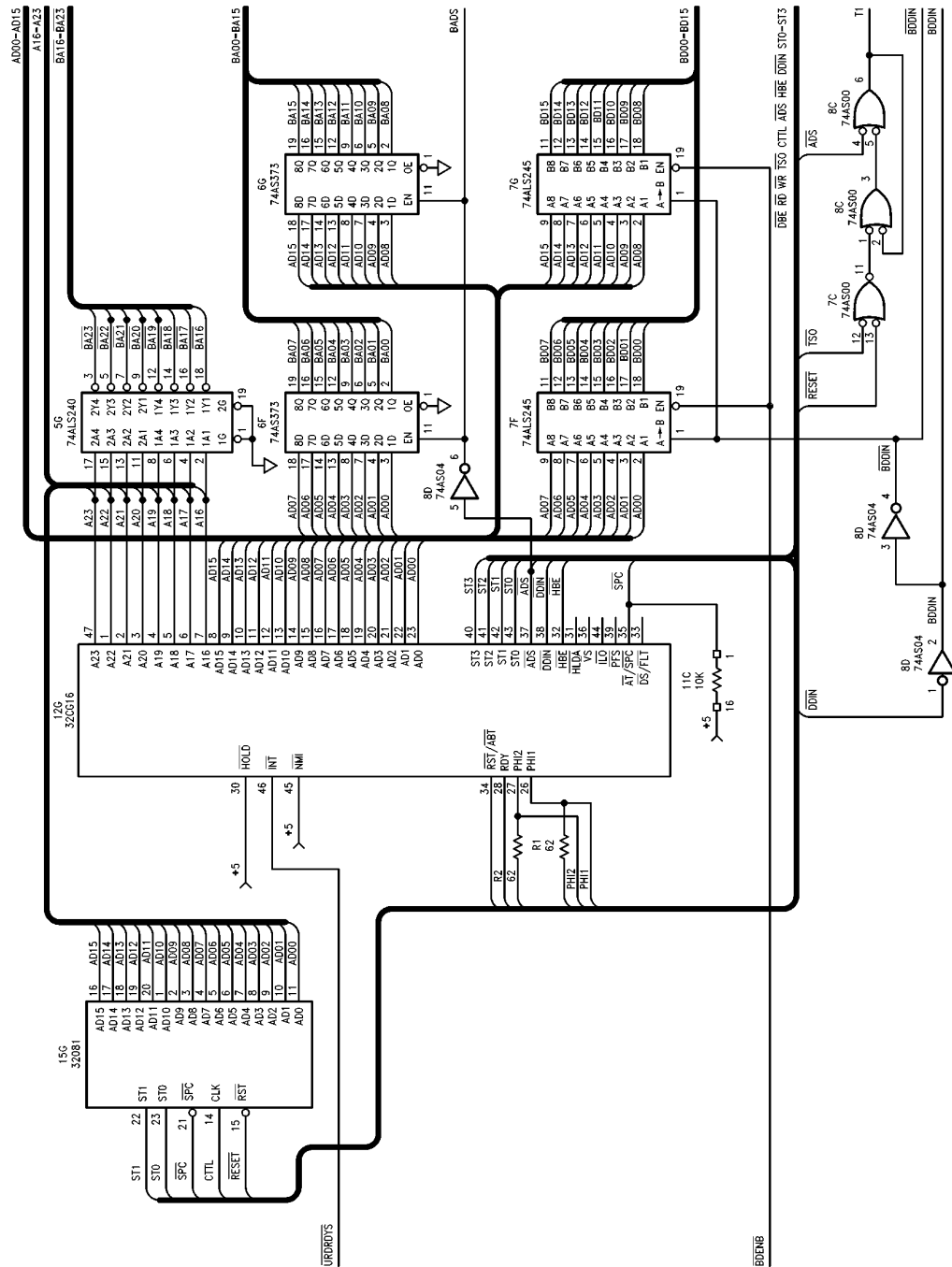
Address Decoders and Timing Control Logic

TL/DD/9976–15

DRAM Address Generation

TL/DD/9976–16

3.1.2 PAL Equations

```
Name        REFRESH.PLD;
Partno      XXXXX;
Date        05/19/87;
Revision    1A;
Designer    FOX;
Company     NSC;
Assembly    X7A;
Location    8B;
Device      p20x10;
/******************************************************************************/
/*                                                                            */
/*  REFRESH:  9 BIT REFRESH COUNTER                                           */
/*                                                                            */
/******************************************************************************/
/*  Allowable Target Device Types: PAL20X10                                   */
/******************************************************************************/
/**  Inputs  **/
Pin    1       = !refresh      ;/* refresh pulse                              */

/**  Outputs  **/
Pin    [15..23]= [ra0..8]      ;/* ram refresh address                        */
Pin    14      = !refron       ;/* refresh enabled output                     */
/**  Declarations and Intermediate Variable definitions **/
$define |       #

/**  Logic Equations  **/
!ra0.d  =       ra0;
!ra1.d  =       !ra1 $ ra0;
!ra2.d  =       !ra2 $ ra0 & ra1;
!ra3.d  =       !ra3 $ ra0 & ra1 & ra2;
!ra4.d  =       !ra4 $ ra0 & ra1 & ra2 & ra3;
!ra5.d  =       !ra5 $ ra0 & ra1 & ra2 & ra3 & ra4;
!ra6.d  =       !ra6 $ ra0 & ra1 & ra2 & ra3 & ra4 & ra5;
!ra7.d  =       !ra7 $ ra0 & ra1 & ra2 & ra3 & ra4 & ra5 & ra6;
!ra8.d  =       !ra8 $ ra0 & ra1 & ra2 & ra3 & ra4 & ra5 & ra6 & ra7;
refron.d=       'b'1;
```

```
Name        RAM.PLD ;
Partno      XXXXX ;
Date        07/25/87 ;
Revision    1A ;
Designer    FOX ;
Company     NSC ;
Assembly    X7A ;
Location    9F ;
Device      p20r8 ;
/********************************************************************************/
/*                                                                            */
/*  RAM CONTROL: HARDWARE RMW BPU CYCLE, SEPARATE BUSES                        */
/*  6/17: Two States of refadr                                                */
/*  6/19: Invert rsl                                                          */
/********************************************************************************/
/*  Allowable Target Device Types: PAL20R8B                                   */
/********************************************************************************/

/**   Inputs  **/

Pin     1   = cttl      ;       /* clock input                                */
Pin     2   = !ddin     ;       /* data direction in signal                   */
Pin     3   = drams1    ;       /* DRAM state counter, bit 1                  */
Pin     4   = drams2    ;       /* DRAM state counter, bit 2                  */
Pin     5   = !bpurmw   ;       /* BPU read modify write cycle                */
Pin     6   = !bpuread  ;       /* BPU source read (comb.)                    */
Pin     7   = !ramsel   ;       /* Any RAM address decode                     */
Pin     8   = busy      ;       /* DRAM busy indication (rsl | refresh)       */
Pin     9   = !acwait   ;       /* Advanced CWAIT from ROM, or I/O            */
Pin     10  =!rsl       ;       /* ram cycle delayed by one Tstate            */
Pin     11  = !srefreq  ;       /* Refresh Request                            */
Pin     14  = tl        ;       /*Processor T1 state                          */
Pin     23  !a23        ;       /* Address 23                                 */

/**   Outputs  **/
Pin     15  = !refresh  ;       /* refresh cycle                              */
Pin     16  = !cwait    ;       /* 32C201 cwait                               */
Pin     17  = !cas      ;       /* CAS, local & cartridge                     */
Pin     18  = !rascart  ;       /* RAS for DRAM cartridge                     */
Pin     19  = !raslcl   ;       /* RAS for local DRAM                         */
Pin     20  = !ramwe    ;       /* DRAM Write enable                          */
Pin     21  = !aramrd   ;       /* DRAM read                                  */
Pin     22  = !pending  ;       /* DRAM cycle requested, but ctl busy         */
min [refresh, cwait, cas, rascart, raslcl, ramwe, aramrd, pending] = 2;
/** Declarations and Intermediate Variable Definitions **/
field waitseq = [pending, cwait];
$define widle    0      /* wait sequencer idle */
$define busywt   3      /* wait sequencer waiting for busy DRAM */
$define cextwt   1      /* wait sequencer waiting for cycle extension */
```

```
field ctl = [refresh,cas,raslcl,rascart];
$define idle    00
$define cras    01
$define crascas 05
$define casend  04
$define lras    02
$define lrascas 06
$define refadr  08
$define refras  0b
$define |       #
field drscount = [drams2..drams1];
/**  Logic Equations  **/
        lcl_sel        = ramsel & !a23;
        cart_sel       = ramsel & a23;

        lclread        = !a23 & ddin;
        lclwrite       = !a23 & !ddin;

        holdoff        rsl;
/*      busy           = refresh | holdoff;   (generated externally)              */
        cart_start     = cart_sel & (tl | pending) & !holdoff;
        local_start    = lcl_sel & (tl | pending) & !holdoff;
        ram_start      = cart_start | local_start;
        drrco          = drscount: [6..7] & ramwe;
sequence waitseq {
/*      acwait & ramsel are mutually exclusive conditions  */
present widle    if (ramsel | bpurmw & bpuread) & busy & tl    next busywt;
                 if acwait | (ramsel & !busy & tl & !bpurmw)
                            next cextwt;
                 default next widle;
present busywt   if busy                         next busywt;
                 if !busy & (bpurmw)             next widle;
                 if !busy & !(bpurmw)            next cextwt;
present cextwt   if ramsel & drscount: [0..1] | acwait   next cextwt;
                 default next widle;
}
sequence ctl {
present idle     if cart_start                   next cras;
                 if local_start                  next lras;
                 if !ram_start & srefreq          next refadr;
                 default next idle;
present cras     if !rsl                         next cras;
                 if rsl                          next crascas;
present crascas  if (!bpurmw & drscount: [4..7]) | (bpurmw & drrco)
                                                 next casend;
                 default next crascas;
present lras                                     next lrascas;
```

16

```
present lrascas if (!bpurmw & drscount: [4..7]) | (bpurmw & drrco)
                                                  next casend;
              default next lrascas;
present casend   if srefreq                   next refadr;
              if !srefreq                     next idle;
present refadr   if srefreq                   next refadr;
              if !srefreq & !rsl              next refras;
              if !srefreq & rsl               next idle;
present refras   if ramwe                     next refadr;
              default next refras;
}
/*  remember ramwe & aramrd are delayed by one t-state  */
ramwe.d = !refresh & (bpurmw & drscount: [6..7] & !ramwe
                        | !bpurmw & !ddin & (ram_start | ctl: cras
                                    | (cart_sel & drscount: [0..3]) | ctl:lras)
                     )
        | ctl:refras & rsl & !ramwe;

aramrd.d = (bpurmw & drscount: [0..3] | !bpurmw & ddin)
         & (ctl:cras | ctl:crascas | ctl:lras | ctl:lrascas);
```

```
Name        DCD1.PLD;
Date        07/03/87;
Revision    1A;
Designer    FOX;
Company     NSC;
Assembly    X7A;
Location    9G;
Device      p1618;
/*******************************************************************************/
/*                                                                           */
/*  DECODE 1: I/O DECODE, PROM & HPC I/F WAIT CONTROL                         */
/*  6/3: two waits for hpc write                                             */
/*  6/4: 1 wait min. for ALL i/o, including HPC                              */
/*  6/4: 3 wait min. for i/o                                                 */
/*                                                                           */
/*******************************************************************************/
/*  Allowable Target Device Types: PAL16L8B                                   */
/*******************************************************************************/
/**   Inputs  **/
Pin        [1..8]    = [a23..16]     ;/* high order address bus              */
Pin        9         = ba8           ;/* address bit 8                       */
Pin        11        = !ddin         ;/* cpu ddin/                           */
Pin        13        = !uwrrdys      ;/* (HPC) UWRRDY/, synchronized         */
Pin        14        = tl            ;/* Tl state of CPU                     */
Pin        17        = !urdrdys      ;/* (HPC) URDRDY/, synchronized         */

/**   Outputs  **/
Pin        12        = !iosel        ;/* I/O select decode                   */
Pin        15        = !waitlo       ;/* WAIT1 output                        */
Pin        16        = !wait2o       ;/* WAIT2 output                        */
Pin        18        = !acwait       ;/* Advance CWAIT for RAM ctl           */
Pin        19        = !ramsel       ;/* DRAM address decode                 */

/**   Declarations and Intermediate Variable Definitions  **/
$define  |           #
field address       = [a23..16]      ;/* address field                      */
field waitv         = [acwait,wait2o,waitlo]; /* wait value field            */
$define nowaits     "b'000
$define waitlv      ("b'100 & tl)
/* note use of # in next 3 defines because $define not nestable              */
$define wait2v                      ("b'101 & ("b'011 # "b'100 & tl))
$define wait3v                      ("b'110 & ("b'011 # "b'100 & tl))
$define wait4v                      ("b'111 & ("b'011 # "b'100 & tl))
$define cwaitonly "b'100

/**   Logic Equations  **/
ramsel              = address: [0780000..07fffff] | address: [0800000..0bfffff];
iosel               = address: [0fd0000..0ffffff] & !ba8;
waitv               = wait3v & address: [0000000..00fffff] /* main rom, 3 waits */
                    | wait4v & address: [0200000..05fffff] /* font rom, 4 waits */
                    | wait3v & address: [0fd0000..0ffffff] & !ba8 /* i/o, 1 wait */
                    | cwaitonly & address: 0ff0000 & !ba8 &
                                    (!urdrdys & ddin | !uwrrdys & !ddin);
```

```
Name        DCD2.PLD;
Partno      XXXXX;
Date        07/27/87;
Revision    1C;
Designer    FOX;
Company     NSC;
Assembly    X7A;
Location    10D;
Device      p20l8;
/*******************************************************************************/
/*                                                                             */
/*  DECODE 2: ROM DECODE, BUFFER CONTROL, BPU DECODE
/*  5/24: included enbpu in bpucyc generation                                  */
/*  5/28: added bpucyc to rdenb                                                */
/*  5/31: added fcxxxx to bdenb                                                */
/*  6/23: added buffer disable term for SPLICE                                 */
/*  7/25: reconfigured for bpurmw & bpuread                                    */
/*  7/27: inverted polarity of enbpu ≥ enablebpu (for master enb)             */
/*******************************************************************************/
/*  Allowable Target Device Types: PAL20B                                      */
/*******************************************************************************/
/**   Inputs  **/
Pin         1       = !ddin       ;/* ddin/ from cpu                           */
Pin         = [2..9]=[a23..16]    ;/* high order address bus                   */
Pin         10      = !enablebpu ;/* BPU enable, static bit                    */
Pin         11      = !bufdis     ;/* buffer disable                           */
Pin         13      = !dbe        ;/* dbe/ from tcu                            */
Pin         14      = !datacyc    ;/* data cycle status decode                 */
Pin         23      = ramcyc      ;/* ram cycle in progress                    */
/**   Outputs  **/
Pin         15      = !bdenb      ;/* BD bus enable                            */
Pin         16      = !romsel     ;/* Main rom select                         */
Pin         17      = !romcart    ;/* rom cartridge select                     */
Pin         18      = !bpurmw     ;/* BPU read modify write                    */
Pin         19      = !bpuread    ;/* BPU read cycle (comb.)                   */
Pin         20      = !vramsel    ;/* video ram select                        */
Pin         21      = rdbufin     ;/* RAM data bus direction (in)              */
Pin         22      = !rdenb      ;/* RAM data bus enable                     */
/**   Declarations and Intermediate Variable Definitions  **/

field address       = [a23..16]  ;/* address field                            */
romspace            = address: [0000000..05fffff];
ramspace            = address: [0780000..0bfffff];
stack               = address: [0780000..078ffff];

$define  |          #
            min b_ddin = 0;
/**   Logic Equations  **/
romsel = address: [0000000..00fffff];   /* main rom                           */
romcart = address: [0200000..05fffff];  /* font rom                           */
```

```
vramsel = address: [0f00000..0f0ffff];                          /* video ram (scan buffer)     */
/*
/*              bpucyc & b_ddin are D latches implemented in the PAL
/*
/*              basic d latch equation (w/o set or clear) is:
/*                          Q = (G & D) | (!G & Q) | (D & Q)
/*
/*              The b_ddin latch is fall through while ramcyc not asserted,
/*              latched while ramcyc is asserted, therefore, for both latches:
*/

                g           = !ramcyc;
/*
/*              The bpurmw latch d input is ""bpurange'', defined as:
*/

                bpurange=   address:   [0000000..05fffff]        /* rom                         */
                          | address: [0790000..0bfffff];         /* dram, less stack            */
/*              This ""d'' input would use too many terms. The bpucyc output,
/*              however, need only be latched when it is asserted, as this is
/*              the situation that can allow the cpu and ram control to
/*              not be synchronized. This simplification allows the simplification
/*              of the latch to:
/*                          Q = D | (!G & Q)
*/

                bpurmw = enablebpu & (!ddin & bpurange & datacyc | (!g & bpurmw));
                bpuread = enablebpu &  ddin & bpurange & datacyc;
/*              rdenb enables cpu access to the ram data bus
*/

                rdenb       = dbe & bufdis &
                              ( !bpurmw & bpuread & romspace        /* buffer must be off for bpu
                                                                    /* but on for source in rom    */
                               /* no DRAM or bpu control writes are permitted                      */
                               /* while in inner loop of bitblt                                    */
                               /* (within interrupt ok due to vector read!)
                               | ramspace
                               | address: [0fe0000..0feffff]);     /* i/o access to bpu           */
                !rdbufin    = (ramspace | address: [0fe0000..0feffff]) & !ddin
                              | romspace & bpuread;
                bdenb       = dbe & !bufdis & (romspace            /* any rom                     */
                                    | address: [0f00000..0f0ffff] /* scan buffer                 */
                                    | address: [0fd0000..0fdffff] /* cmnd/status                 */
                                    | address: [0ff0000..0fffffff] /* non-bpu i/o                 */
```

3.2 Application Connections

The connections made to the HPC are shown in schematic sheets 2 through 4.

3.2.1 LCD Data

An 8-bit parallel interface connects the upper half of Port A, through buffers and latches on Sheet 4, to a Hitachi HD44780 alphanumeric LCD display controller. The signals in our application are inverted with respect to the HD44780 documentation, due to the nature of the front panel module we used.

Sending data from the HPC to the LCD display involves the following procedure:

1. Setup the $\overline{RS}$ signal: 1 for a command, 0 for data.
   This is done by setting up LCD Contrast status on the high-order byte of Port A (pins A8–A15), with the desired $\overline{RS}$ state on pin A11, then pulsing the signal LCVCLK (pin B9) high, the low.

2. Setup the panel data on HPC pins A8–A15.

3. Set the PNLCLK signal (pin B7) low for 1.2 μs, then high. This clocks the data into the LCD display controller. Note that the latch in area B6 of Sheet 4 is effectively serving only as a buffer; the PNLCLK Enable signal, being normally high, allows data to fall through whenever it changes when used as described here.

4. Since the handshaking capability of the HD44780 is not being used here, it is necessary for the HPC to use an internal timer to determine when the controller is ready after sending a command or data. The delay time is either 120 μs or 4.9 ms, depending on the type of command sent.

3.2.2. LCD Contrast (LCD Voltage)

A three-bit value is presented for LCD contrast on signals $\overline{CTRST0}$ through $\overline{CTRST2}$. A value of 000 is highest contrast, and 111 is lowest contrast. To change the contrast, the value is placed on HPC pins A8 (LSB), A9 and A10 (MSB), the LCVCLK (pin B9) is pulsed high, then low.

Note that some other bits within this latch have other functions: bit 3 (from HPC pin A11) is the $\overline{RS}$ signal to the LCD controller, and bit 7 (from pin A15) is used by the HPC firmware as a Fatal Error flag. These bits must be setup correctly whenever the LCD Contrast latch is written to.

3.2.3 LEDs

Up to 8 LED indicators may be connected, through the latch in area A6 of Sheet 4, to the upper byte of Port A. The LED's are assumed to be connected already to their own current-limiting resistors.

The desired data is setup on Port A pins A8–A15, then a pulse is presented on the LEDCLK signal (pin B14); high and then low. Data is presented in complemented form by the HPC (0 = on, 1 = 0ff). Any or all (or none) of the latch bits may be connected to drive LEDs.

3.2.4 Speaker (Beeper)

A tone is produced on a speaker by enabling Port P pin P3 as the Timer T7 output, and running Timer T7 so as to produce a 3 kHz square wave. Since timer outputs toggle on underflows, this corresponds to a timer underflow rate of 6 kHz. The tone signal is shown is area D1 of Sheet 2.

3.2.5 Pushbutton Switches

Up to eight pushbuttons may be connected to the HPC's Port D pins, through the buffer in area D6 of Sheet 3. Each pushbutton is assumed to be an SPST switch, shorting to ground when depressed. The pull-up resistors present a ''1'' level otherwise. The HPC must de-bounce the inputs in its firmware before issuing them to the CPU.

The pushbuttons are examined every 10 ms, by setting the $\overline{ENASTTS}$ signal (pin B13) low while ensuring that $\overline{ENCDATA}$ (pin B12) is high. This presents the switch outputs onto Port D. Unused bits should be pulled high to avoid triggering spurious pushbutton events.

3.3 Protocol Between CPU and HPC

The scheme supported by the UPI Driver program is asynchronous full-duplex communication with CPU. That is, either side is allowed to speak at any time. To avoid confusion, however, any message is restricted to send data in only one direction: in sequences initiated by the CPU (''Command'' sequences), only the CPU talks, and in sequences initiated by the HPC (''Interrupt'' sequences), only the HPC talks. Thus, a Command sequence and an Interrupt sequence can be in progress simultaneously without confusion.

Acknowledgement of a Command or an Interrupt sequency is possible; a Command can trigger an acknowledgement Interrupt sequence, and an Interrupt sequence can result in a subsequent Command sequence. The critical distinction, though, is that the acknowledgement need not come immediately. If, for example, the HPC is already in the process of sending an Interrupt message, and receives a Command, it will complete the current Interrupt sequence before acknowledging the Command with a new Interrupt.

Command sequences (from the CPU to the HPC) consist of a one-byte command code, followed by any argument values necessary to complete the command. Each byte written to the HPC triggers an internal interrupt (I3); the HPC buffers up these bytes until a full command has been received, then acts on it in the last byte's interrupt service routine. Commands taking a significant amount of processing time can be scheduled within the HPC using interrupts, either from external events or from one of the HPC's eight timers; each interrupt triggering the next step of the command.

Interrupt sequences (from the HPC to the CPU) operate similarly, but with a small difference. Only the first byte presented by the HPC causes an interrupt to the CPU; this byte is the interrupt vector value, which triggers the interrupt (through the $\overline{RDRDY}$ pin) and selects the CPU's service routine. The CPU remains in its interrupt service routine until the transfer of data associated with that interrupt event is finished, then returns to its previous task. This is not to say that the CPU must keep all other interrupts disabled during an Interrupt sequence, but only that no other interrupt occurring during this time may cause the CPU to read from the HPC, or to terminate reading, until the current Interrupt sequence is complete. With the NS32C016 processor as host, the main challenge is to keep the Interrupt Acknowledge bus cycles from other interrupts, which appear as Read cycles, from causing $\overline{URD}$ pulses to the HPC. It is possible to distinguish a Non-Maskable Interrupt from a Maskable Interrupt by the address asserted by the CPU in acknowledging the interrupt, and in a larger kind of system containing an NS32202 Interrupt Control Unit, the NS32000 Cascaded Interrupt feature can be used to prevent unwanted reads from the HPC from occurring as a result of other Maskable interrupts as well. In our application hardware, the only type of extraneous interrupt occurring is the Non-Maskable Interrupt; address decoding logic isolates the HPC's UPI port from these.

21

3.4 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. Unless 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 command) enables the !RTC and !BUTTON-DATA interrupts. The INITIALIZE command may be re-issued by the CPU 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	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.
02	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.

03	SEND-LED	The single 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.

04	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.
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 disable during this time. It also must not enter a command byte during this time because the byte may be lost.

3.5 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".

00–0F (Reserved for CPU internal traps and the NMI interrupt.)

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.
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.

| 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:

x	x	x	F	x	x	C	N

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	(Reserved for COMMAND)						

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 *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 HPI IBUF register for a few milliseconds before the HPC starts to process it.

4.0 SOURCE LISTINGS AND COMMENTARY

4.1 HPC Firmware Guide

Refer to this section for help in following the flow of the HPC firmware in the listing below. Positions in the code are referenced by assembly language labels rather than by page or line numbers.

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 an 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 was done for ease of breakpointing, to keep this routine at a constant address as changes were made elsewhere in the firmware.

4.1.1 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 usually called as a subroutine (although it never returns). It disables HPC internal interrupts, and then sets bit 7 of the LCD Contrast Latch as a trigger for a logic analyzer, MOLE or ISE system.

Its next action is to display its subroutine return address in hexadecimal on the LCD panel. This address shows where the error was detected. The HPC then enters an infinite loop, which continuously presents the !DIAG interrupt. It may be terminated either by a hardware reset or by sending the RESET command from the CPU. On receiving the RESET command, the HPC jumps to label "xreset", which is within the command processing routine. The "xreset" rou-

tine waits for the CPU to read from the UPI port, then clears a set of registers to simulate a hardware reset and jumps to the start of the program.

4.1.2 Initialization

On receiving a Reset signal, the HPC begins execution at the label "start". A required part of any application is to load the PSW register, to select the desired number of Wait states (without this step, the Reset default is 4 Wait states, which is safe but usually unnecessary).

Other initializations here are application-dependent, and so they relate to our application system and front-panel operations.

At label "srfsh", the program starts the Refresh clock pulses running for the dynamic RAM on our application hardware, from HPC pin P0 (controlled by Timer T4). For debugging purposes, a circuit within the RAM controller section performs continuous refreshes during Reset pulses, so data in dynamic RAM is never lost unless power is removed.

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. This is done for debugging purposes, to help ensure that programming errors involving uninitialized data will have more consistent symptoms.

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. This is not necessary in all applications, since the Stack Pointer starts at address 0002, but for our purposes it was more convenient to relocate it.

At "tminit", the timers T1–T3 are stopped and any interrupts pending from timers T0–T3 are cleared.

In addition, some miscellaneous port initializations are performed here. The upper byte of Port A is set as an output port (for data going to the LCD and LED displays), and the Port B pins which select pushbutton data are initialized.

At "sled", the LED control signals are initialized, and all LED indicators on the panel 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. (Timer T4 keeps running for dynamic RAM refresh.)

At "sled", the panel 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", located at the end of the program, which simply writes the character in the accumulator out to the LCD display and waits for approximately 10 ms. Note that if the CPU fails to initialize the LCD display further, a single cursor (underscore) character is all that appears: a recognizable symptom of a CPU problem.

The program now continues to label "minit", which performs some variable initializations which are necessary for operation of the UPI Driver itself (as opposed to the application). This much must always be present, but any other initializations required by the application should appear here as well. For our front-panel application, there are no such initializations required.

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 loop at label "mainlp".

4.1.3 Main Program (UPI Output to CPU)

The Main Program is the portion of the UPI Driver that runs with interrupts enabled. It consists of a scanning loop at label "mainlp", calling 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. The main loop scans a bit-mapped variable in on-chip RAM that is set up by interrupt service routines (a word called "alert") 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. Each subroutine first checks whether the UPI interface's OBUF register is empty, and if not, it waits (by calling the subroutine "rdwait"). It then writes the 32000 interrupt vector number to the OBUF register. This has the effect of interrupting the CPU (Because the pin $\overline{URDRDY}$ 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 the data value taken as the vector number. (Note, however, that a Non-Maskable interrupt is allowed. It simply delays the process of reading data from the HPC. Since the HPC is running its main program at this point, with its internal interrupts still enabled, it is not stalled by this situation.)

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.

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".)

4.1.4 Interrupt Service Routines

All of the remaining routines are entered by the occurrence of an interrupt.

4.1.4.1 UPI Port Input from CPU (Interrupt I3)

This interrupt service routine, at label "upiwr", accepts commands from the CPU. Each byte of a command triggers an interrupt on the I3 pin. When the last byte is received, the command is processed before the I3 interrupt routine returns. The HPC is therefore immediately ready to start collecting another command.

Any command that involves waiting is only initiated before the I3 routine returns, and interrupts are set up to activate more processing when the time is right. Therefore, this interrupt service routine returns promptly, even for time-consuming commands.

At any time, the "upiwr" routine may be in one of the following states:

1. Waiting for the first byte of a command. In this state, the variable "curcmd" (Current Command) has its top bit ("cmdemp") set, meaning that it is empty. When a byte is received from the CPU in this state, this routine jumps to the label "firstc". The byte is placed in the "curcmd" byte (clearing the top bit), and then a multi-way branch (jidw) is performed, whose destination depends on the contents of the byte. The possible destinations have labels starting with the letters "fc". If the command has only one byte (for example, the command BEEP), it is processed immediately in the "fc" sequence, and the "curcmd" variable is set empty again. If, however, the command is longer than one byte, its "fc" routine will place a value into the variable "numexp", which gives the number of additional bytes that are expected for this command, and will return from the interrupt. Note that the "curcmd" byte now appears to be full, because its top bit is no longer set.

2. Collecting bytes of a command. The code that is relevant in this state is between the labels "upiwr" and "lastc". This state is in effect while the "cmdemp" bit of "curcmd" is zero and the "numexp" variable is non-zero. Each I3 interrupt causes the routine to place the command byte into a buffer ("cpubuf", with pointer variable "cpuad"), decrement the "numexp" variable, and return if the result is non-zero. If the result is zero, then the routine has collected an entire command, and it goes to the label "lastc", and enters state (3) below.

3. In this state, the requested number of bytes has been collected, and this usually means that the entire command, except for the first byte, is in the "cpubuf" area of RAM. The code for this state is at label "lastc". First, the "curcmd" byte is checked to see whether "extended collection" is being performed (bit 6 set: see below). If not, the "curcmd" byte is set empty. A multiway branch is then performed (jidw), which transfers control depending on the command byte in "curcmd". All routines that are destinations of this branch start with the letters "lc". The "lc" routine for each command uses the data in "cpubuf" to process the current command. In some cases, this processing is completed very quickly. For example, at label "lcsled", a value is simply transferred from "cpubuf" to a latch that drives the LEDs on the front panel, and this interrupt service routine returns. But a more complex command can move data out of "cpubuf" to other variables in RAM, and start a timer to sequence the process of executing the command.

In some commands (for example, SEND-LCD), state (3) above is entered twice. This is called "extended collection", and occurs when a command has variable length. State (3) is entered once to collect enough information to determine the exact length of the command. It then sets up the "numexp" variable again, re-entering state (2) to collect the remainder of the command. When state (3) is entered the second time, it processes the command. A bit in the "curcmd" variable (bit 6, called "getcnt") is set in state (1), which indicates that another collection will be performed, and prevents state (3) from setting the "curcmd" byte empty the first time it is entered.

Command Processing Routines

INITIALIZE	I3 interrupt labels:	State 1 = fcinit	State 3 = lcinit

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 uses the "extended collection" feature. At label "fcslcd", two bytes are requested for collection, but the "getcnt" bit of "curcmd" is set, meaning that these are not the last bytes of the command. At label "lcslcd" (jumping to label "lcslc1"), the length of the instruction is determined from the #bytes value supplied by the CPU, and a second collection of bytes is requested, this time with the "getcnt" bit off. When the last byte has been collected, control is transferred to the label "lcslcd", then to "lcslc2". Here, the data bytes for the panel are unloaded from the CPU buffer area "cpubuf" into the LCD string buffer "lcdbuf". The flag (RS) bits are loaded into variable "lcdsfg", and the number of bytes to be sent to the LCD display is placed into variable "lcdsct". Timer T6 is now started, to provide scheduling interrupts for writing the bytes from the LCD string buffer to the LCD display.

On occurrence of each T6 interrupt (labels "t6int" and "t6nxtc"), one byte is written to the LCD display. Depending on the state of the RS flag for that byte, and the value sent to the panel, T6 may run for either 120 μs or 4900 μs before it triggers the next transfer. When the last character has been transferred, and Timer T6 has provided the proper delay after it, the bit "alcdak" is set in the "alert" word, requesting the main program to send an !ACK-SEND-LCD interrupt to the CPU.

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

At label "lcsled", the byte provided by the CPU is written to the LED latch.

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

At label "fcbeep", Port P pin P3 is enabled to toggle on each underflow of Timer T7, which has been initialized at the beginning of the program (label "stmrs") to underflow at a rate of 6 kHz. Pin P3, then, presents a 3 kHz square wave to the panel buzzer. To time out the duration of the beep tone, interrupts from Timer T0 are enabled, which then occur once every 53 ms. The variable "beepct" is set up with the number of T0 interrupts to accept, and is decremented on each T0 interrupt. When it has been decremented to zero (meaning that one second has elapsed), pin P3 is reset to a constant zero to turn off the tone.

4.1.4.2 Background Timer (T1) Task

The Timer T1 interrupt service routine represents a task that is not triggered directly by CPU commands. Its functions are to interrupt the CPU periodically for the Real-Time Clock function, and to present the !BUTTON-DATA interrupt whenever the pushbutton inputs change state.

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", to "t1poll", to "t1int"), then if the real-time interrupt is enabled, the variable "rtccnt" is decremented to determine whether an !RTC interrupt should be issued to the CPU. If so, the bit "artc" in the "alert" word is set, requesting the main program to issue the interrupt. The main program, at label "sndrtc", actually interrupts the CPU. No other data is passed to the CPU with the interrupt.

At label "kbdchk" the panel pushbutton switches are also sampled. If the pattern matches the last sample taken (saved in variable "swlast") then it is considered to be sta-ble, and it is then compared to the last switch pattern sent to the CPU (in variable "swlsnt"). If the new pattern differs, then it is placed in "swlsnt", and the bit "abutton" in variable "alert" is set, requesting the main program to send a !BUTTON-DATA interrupt. The main program, at label "sndbtn", triggers the interrupt and passes the new pattern to the CPU from variable "swlsnt".

4.1.4.3 Timer T6 Interrupt

Because the LCD controller's command acknowledgement capability was not used in our application, Timer T6 is used to time out the LCD controller's processing times. See the description of the SEND-LCD command above.

4.1.4.4 Timer T0 Interrupt

The interrupt service routine for Timer T0 (labels "tmrint", to "t0poll", to "t0int") is used simply to provide timing for the duration of the speaker tone. The interrupt is enabled in response to the BEEP command from the CPU, and is disabled on occurrence of the interrupt. It provides an interval of approximately one second.

4.2 HPC Firmware Listing

```
NSC ASMHPC, Ver D1-BetaSite (Sep 14 14:30 1987)          HPCUPI                    25-Feb-88 10:05
                                                                                        PAGE   1


     1
     2                                    .title  HPCUPI,'UPI PORT INTERFACE DEMO'
     3
     4                       ;    Demo program for HPC46083 UPI Port:
     5                       ;           Demonstrates use of the HPC as an interface
     6                       ;           between an NS32C016 CPU and some typical
     7                       ;           front-panel types of devices:
     8                       ;                   LED indicators (up to 8)
     9                       ;                   Pushbuttons (up to 8)
    10                       ;                   LCD alphanumeric display controller (Hitachi HD44780)
    11                       ;                   Speaker for error beeps
    12                       ;    Also generates Real-Time Clock interrupts at a
    13                       ;        selectable rate.
    14                       ;
    15                       ;    Generates !DIAG interrupt on errors;
    16                       ;           severity code of NOTE (e.g. real-time event lost),
    17                       ;                   or FATAL (e.g. bad command).
    18                       ;    Recovery from fatal errors provided by RESET command.
    19
    20
```

TL/DD/9976–17

```
 21                                  .form    'Declarations:  Register Addresses'
 22
 23 00C0                     psw     =       x'C0:w  ; PSW register
 24 00C8                     al      =       x'C8:b  ; Low byte of Accumulator.
 25 00C9                     ah      =       x'C9:b  ; High byte of Accumulator.
 26 00CC                     bl      =       x'CC:b  ; Low byte of Register B.
 27 00CD                     bh      =       x'CD:b  ; High byte of Register B.
 28 00CE                     xl      =       x'CE:b  ; Low byte of Register X.
 29 00CF                     xh      =       x'CF:b  ; High byte of Register X.
 30
 31 00D0                     enir    =       x'D0:b
 32 00D2                     irpd    =       x'D2:b
 33 00D4                     ircd    =       x'D4:b
 34 00D6                     sio     =       x'D6:b
 35 00D8                     porti   =       x'D8:b
 36 00E0                     obuf    =       x'E0:b  ; (Low byte of PORTA.)
 37 00E1                     portah  =       x'E1:b  ; High byte of PORTA.
 38 00E2                     portb   =       x'E2:w
 39 00E2                     portbl  =       x'E2:b  ; Low byte of PORTB.
 40 00E3                     portbh  =       x'E3:b  ; High byte of PORTB.
 41 00E6                     upic    =       x'E6:b
 42 00F0                     ibuf    =       x'F0:b  ; (Low byte of DIRA.)
 43 00F1                     dirah   =       x'F1:b  ; High byte of DIRA.
 44 00F2                     dirb    =       x'F2:w
 45 00F2                     dirbl   =       x'F2:b  ; Low byte of DIRB.
 46 00F3                     dirbh   =       x'F3:b  ; High byte of DIRB.
 47 00F4                     bfun    =       x'F4:w
 48 00F4                     bfunl   =       x'F4:b  ; Low byte of BFUN.
 49 00F5                     bfunh   =       x'F5:b  ; High byte of BFUN.
 50
 51 0104                     portd   =       x'0104:b
 52 0120                     enu     =       x'0120:b
 53 0122                     enui    =       x'0122:b
 54 0124                     rbuf    =       x'0124:b
 55 0126                     tbuf    =       x'0126:b
 56 0128                     enur    =       x'0128:b
 57
 58 0140                     t4      =       x'0140:w
 59 0142                     r4      =       x'0142:w
 60 0144                     t5      =       x'0144:w
 61 0146                     r5      =       x'0146:w
 62 0148                     t6      =       x'0148:w
 63 014A                     r6      =       x'014A:w
 64 014C                     t7      =       x'014C:w
 65 014E                     r7      =       x'014E:w
 66 0150                     pwmode  =       x'0150:w
 67 0150                     pwmdl   =       x'0150:b        ; Low byte of PWMODE.
 68 0151                     pwmdh   =       x'0151:b        ; High byte of PWMODE.
 69 0152                     portp   =       x'0152:w
 70 0152                     portpl  =       x'0152:b        ; Low byte of PORTP.
```

```
 71 0153                     portph  =       x'0153:b        ; High byte of PORTP.
 72 015C                     eicon   =       x'015C:b
 73
 74 0182                     t1      =       x'0182:w
 75 0184                     r1      =       x'0184:w
 76 0186                     r2      =       x'0186:w
 77 0188                     t2      =       x'0188:w
 78 018A                     r3      =       x'018A:w
 79 018C                     t3      =       x'018C:w
 80 018E                     divby   =       x'018E:w
 81 018E                     divbyl  =       x'018E:b        ; Low byte of DIVBY.
 82 018F                     divbyh  =       x'018F:b        ; High byte of DIVBY.
 83 0190                     tmmode  =       x'0190:w
 84 0190                     tmmdl   =       x'0190:b        ; Low byte of TMMODE.
 85 0191                     tmmdh   =       x'0191:b        ; High byte of TMMODE.
 86 0192                     t0con   =       x'0192:b
 87
 88
```

```
 89                                .form    'Declarations:  Register Bit Positions'
 90
 91                        ; Name       Position      Register(s)
 92                        ; ----       --------      -----------
 93
 94 0000                   gie     =       0         ; enir
 95 0002                   i2      =       2         ; enir, irpd, ircd
 96 0003                   i3      =       3         ; enir, irpd, ircd
 97 0004                   i4      =       4         ; enir, irpd, ircd
 98 0005                   tmrs    =       5         ; enir, irpd
 99 0006                   uart    =       6         ; enir, irpd
100 0007                   ei      =       7         ; enir, irpd
101
102 0001                   uwmode  =       1         ; ircd
103 0000                   uwdone  =       0         ; irpd
104
105 0000                   tbmt    =       0         ; enu
106 0001                   rbfl    =       1         ; enu
107 0004                   b8or9   =       4         ; enu
108 0005                   xbit9   =       5         ; enu
109 0002                   wakeup  =       2         ; enur
110 0003                   rbit9   =       3         ; enur
111 0006                   frmerr  =       6         ; enur
112 0007                   doeerr  =       7         ; enur
113 0000                   eti     =       0         ; enui
114 0001                   eri     =       1         ; enui
115 0002                   xtclk   =       2         ; enui
116 0003                   xrclk   =       3         ; enui
117 0007                   b2stp   =       7         ; enui
118
119 0000                   wrrdy   =       0         ; upic
120 0001                   rdrdy   =       1         ; upic
121 0002                   la0     =       2         ; upic
122 0003                   upien   =       3         ; upic
123 0004                   b8or16  =       4         ; upic
124
125 0000                   t0tie   =       0         ; tmmdl
126 0001                   t0pnd   =       1         ; tmmdl
127 0003                   t0ack   =       3         ; tmmdl
128 0004                   t1tie   =       4         ; tmmdl
129 0005                   t1pnd   =       5         ; tmmdl
130 0006                   t1stp   =       6         ; tmmdl
131 0007                   t1ack   =       7         ; tmmdl
132 0000                   t2tie   =       0         ; tmmdh
133 0001                   t2pnd   =       1         ; tmmdh
134 0002                   t2stp   =       2         ; tmmdh
135 0003                   t2ack   =       3         ; tmmdh
136 0004                   t3tie   =       4         ; tmmdh
137 0005                   t3pnd   =       5         ; tmmdh
138 0006                   t3stp   =       6         ; tmmdh
```

TL/DD/9976-20

```
139 0007                  t3ack   =       7       ; tmmdh
140
141 0000                  t4tie   =       0       ; pwmdl
142 0001                  t4pnd   =       1       ; pwmdl
143 0002                  t4stp   =       2       ; pwmdl
144 0003                  t4ack   =       3       ; pwmdl
145 0004                  t5tie   =       4       ; pwmdl
146 0005                  t5pnd   =       5       ; pwmdl
147 0006                  t5stp   =       6       ; pwmdl
148 0007                  t5ack   =       7       ; pwmdl
149 0000                  t6tie   =       0       ; pwmdh
150 0001                  t6pnd   =       1       ; pwmdh
151 0002                  t6stp   =       2       ; pwmdh
152 0003                  t6ack   =       3       ; pwmdh
153 0004                  t7tie   =       4       ; pwmdh
154 0005                  t7pnd   =       5       ; pwmdh
155 0006                  t7stp   =       6       ; pwmdh
156 0007                  t7ack   =       7       ; pwmdh
157
158 0000                  t4out   =       0       ; portpl
159 0003                  t4tfn   =       3       ; portpl
160 0004                  t5out   =       4       ; portpl
161 0007                  t5tfn   =       7       ; portpl
162 0000                  t6out   =       0       ; portph
163 0003                  t6tfn   =       3       ; portph
164 0004                  t7out   =       4       ; portph
165 0007                  t7tfn   =       7       ; portph
166
167 0000                  eipol   =       0       ; eicon
168 0001                  eimode  =       1       ; eicon
169 0002                  eiack   =       2       ; eicon
170
171 0005                  so      =       5       ; portbl, dirbl, bfunl
172 0006                  sk      =       6       ; portbl, dirbl, bfunl
173 0007                  pnlclk  =       7       ; portbl, dirbl
174
175 0001                  lcvclk  =       1       ; portbh, dirbh
176                       ; ua0 would be 2  , but requires no setup.
177 0003                  uwrrdy  =       3       ; portbh, dirbh, bfunh
178 0004                  cdata   =       4       ; portbh (enables non-pushbutton data to Port D).
179 0005                  astts   =       5       ; portbh (enables pushbutton data to Port D).
180 0006                  ledclk  =       6       ; portbh, dirbh
181 0007                  urdrdy  =       7       ; portbh, dirbh, bfunh
182
183                       ;       CONSTANTS
184                       ;
185 0011                  xon=    x'11    ; XON character:   Control-Q
186 0013                  xoff=   x'13    ; XOFF character:  Control-S
187
188
```

TL/DD/9976–21

```
     189                                    .form   'Space Declarations'
     190 0000                               .sect   DSECT,BASE,REL  ; Basepage RAM variables (addresses 0000-00BF)
     191
     192                            ; WORD-ALIGNED
     193 0000                       dummy:  .dsw 1  ; x'00,01      ; Destroyed on reset (address 0).
     194 0000                               .set    upicsv,dummy   ; Temporary image of UPIC register.
     195 0002                       alert:  .dsw 1  ; Alert status bits to main program:
     196                                            ;   generate interrupts to CPU.
     197 0003                               .set    alerth,alert+1:b       ; Declare top byte of ALERT word.
     198 0004                       cpuad:  .dsw 1  ; Current address within CPU command buffer.
     199 0006                       cpubuf: .dsw 4  ; Buffer for accepting command parameters from CPU.
     200 000E                       lcdsix: .dsw 1  ; Pointer into LCD character string buffer.
     201
     202                            ;BYTE-ALIGNED
     203 0010                       curcmd: .dsb 1  ; Current command byte from CPU being processed.
     204 0011                       numexp: .dsb 1  ; Number of parameter bytes expected before command processing
     205                                            ;   begins.
     206 0012                       lcvs:   .dsb 1  ; Image of LCD Voltage (Contrast) latch setting;  needed with
     207                                            ;   LCD RS (PAUX0) signal coming from this latch.
     208 0013                       lcdfgs: .dsb 1  ; Holds flag bits for characters sent to Panel LCD display.
     209 0014                       lcdnum: .dsb 1  ; Number of characters to be sent to LCD display.
     210 0015                       lcdsfg: .dsb 1  ; Flag bits associated with characters in LCD String Buffer.
     211 0016                       lcdsct: .dsb 1  ; Counter for characters being sent to LCD display from String
     212                                            ;   Buffer.
     213 0017                       swlast: .dsb 1  ; Last-sampled switch values.
     214 0018                       swlsnt: .dsb 1  ; Last switch values sent to CPU.
     215 0019                       beepct: .dsb 1  ; Beep duration count.  Counts occurrences of T0 interrupt.
     216 001A                       rtcivl: .dsb 1  ; Real-Time Clock Interval (units of 10 milliseconds).
     217 001B                       rtccnt: .dsb 1  ; Real-Time Clock Current Count (units of 10 milliseconds).
     218 001C                       rtevs:  .dsb 1  ; Events to check for on Timer T1 interrupts.
     219 001D                       dsevc:  .dsb 1  ; Diagnostic Interrupt:  Severity Code.
     220 001E                       derrc:  .dsb 1  ; Diagnostic Interrupt:  Error Code.
     221 001F                       dbyte:  .dsb 1  ; Diagnostic Interrupt:  Error Byte.
     222 0020                       dccmd:  .dsb 1  ; Diagnostic Interrupt:  Current Command.
     223 0021                       dqual:  .dsb 1  ; Diagnostic Interrupt:  Qualifier (Command Status).
     224
     225
     226                            ;       BIT POSITIONS
     227
     228
     229                                    ; ALERT status word (low-order byte) bits:
     230
     231 0000                       abutton =       0       ; Pushbutton switch state change.
     232 0001                       artc    =       1       ; Real-Time Interrupt detected.
     233 0002                       adiag   =       2       ; Diagnostic interrupt.
     234 0003                       alcdak  =       3       ; LCD Panel Write Acknowledge.
     235                            ; (Other bits not defined.)
     236
     237                                    ; ALERT status word (high-order byte, named alerth) bits:
     238
```

```
     239                            ; (Other bits not defined.)
     240
     241
     242                                    ; CURCMD byte:  Current CPU command.  The lower 5 bits contain the
     243                                    ;                 command code.  The upper two bits contain
     244                                    ;                 further information about command collection:
     245 0007                       cmdemp= 7       ; Bit 7 (MSB) of curcmd = 1 means that no command is being
     246                                            ;  processed and curcmd byte is "empty".
     247 0006                       getcnt= 6       ; Bit 6 of curcmd = 1 means that the count is being received
     248                                            ;  for a variable-length command.
     249
     250                                    ; LCVS byte:  LCD Voltage (Contrast) Latch memory image.
     251                                    ;             Contains voltage value in its least-significant 3 bits,
     252                                    ;             RS signal to LCD controller in bit 3, and debugging
     253                                    ;             information in its top 4 bits.
     254 0003                       pnlrs=  3       ; Bit 3 is (inverted) RS signal to panel.
     255
     256
     257                                    ; RTEVS byte:  Events to check for at 10-millisecond intervals.
     258                                    ;              (T1 Underflows)
     259 0000                       rtcenb= 0       ; 1 = Real-Time Clock interrupts enabled to CPU.
     260
     261
     262 0000                               .sect   STACK,RAM16,REL         ; On-chip RAM in addresses 01C0-01FF.
     263 0000                       stackb: .dsw    16      ; Space for 8 words beyond
     264                                            ;  interrupt context.
     265 0020                       avail:  .dsw    12      ; Spare portion of this space.
     266 0038                       lcdbuf: .dsw    4       ; LCD String Buffer.
     267
```

```
268                                       .form   'Code Section'
269 0000                                   .sect   CSECT,ROM16,REL ; Code space.  (On-chip ROM)
270
271                                       ; Declarations of subroutines called by one-byte JSRP instruction.
272
273 0000                                   .spt    rdwait             ; Waits for CPU to read a value from UPI port.
274 0000                                   .spt    wrpnl              ; Writes to LCD panel (for initialization only).
275
276                                       ; Program starts at label "start" on reset.  This routine is the fatal
277                                       ; error handler, located here for convenience in setting breakpoint.
278
279 0000 96D018          hangup: rbit    gie,enir           ; Fatal error:  signal it and halt.
280 0003 96120F      R           sbit    7,lcvs             ; Signal error on most-significant bit of
281                                                           ; LCD Contrast Latch.
282 0006 96120B      R           sbit    pnlrs,lcvs         ; Select command mode for LCD controller.
283 0009 8C12E1      R           ld      portah,lcvs        ; Place error on Port A for latch.
284 000C 96E309                  sbit    lcvclk,portbh      ; Clock LCD Contrast Latch high,
285 000F 96E319                  rbit    lcvclk,portbh      ;  then low to load it.
286 0012 B601510A                sbit    t6stp,pwmdh        ;
287 0016 B6015118                rbit    t6tie,pwmdh        ; Set up Timer T6 for non-interrupt use.
288 001A 40                      nop
289 001B B6015119                rbit    t6pnd,pwmdh        ; Clear Pending bit.
290 001F 3F00                    pop     0.w                ; Get error address from stack.
291 0021 B70000C4    R           ld      sp,#stackb         ; In case of stack underflow, re-initialize SP.
292 0025 9001                    ld      A,#x'01
293 0027 2E          R           jsrl    wrpnl              ; Clear LCD panel.
294 0028 96121B      R           rbit    pnlrs,lcvs         ; Set up panel for data.
295 002B 8C12E1      R           ld      portah,lcvs        ; Place error on Port A for latch.
296 002E 96E309                  sbit    lcvclk,portbh      ; Clock LCD Contrast Latch high,
297 0031 96E319                  rbit    lcvclk,portbh      ;  then low to load it.
298 0034 8801                    ld      A,1.b              ; Process first character of return address.
299 0036 3B                      swap    A
300 0037 990F                    and     A,#x'0F
301 0039 A6007AC888  R           ld      A,hextab[A].b
302 003E 2E          R           jsrl    wrpnl              ; Display it on LCD panel.
303 003F 8801                    ld      A,1.b              ; Process second character of return address.
304 0041 990F                    and     A,#x'0F
305 0043 A6007AC888  R           ld      A,hextab[A].b
306 0048 2E          R           jsrl    wrpnl              ; Display it on LCD panel.
307 0049 8800                    ld      A,0.b              ; Process third character of return address.
308 004B 3B                      swap    A
309 004C 990F                    and     A,#x'0F
310 004E A6007AC888  R           ld      A,hextab[A].b
311 0053 2E          R           jsrl    wrpnl              ; Display it on LCD panel.
312 0054 8800                    ld      A,0.b              ; Process last character of return address.
313 0056 990F                    and     A,#x'0F
314 0058 A6007AC888  R           ld      A,hextab[A].b
315 005D 2E          R           jsrl    wrpnl              ; Display it on LCD panel.
316
317 005E 96E611          hgupi:  ifbit   rdrdy,upic         ; Check to see if OBUF register is full.
```

TL/DD/9976-24

```
318 0061 971DE0                     ld       obuf,#vdiag     ; If not, fill it with !DIAG vector
319                                                          ;  continuously.
320 0064 96D213                     ifbit    i3,irpd         ; Check for UPI data ready.
321 0067 41                         jp       hgupi1
322 0068 6A                         jp       hgupi
323
324 0069 82A5F0DC        hgupi1: ifeq       ibuf,#x'A5      ; Check for RESET command.
325 006D 41                         jp       hgrst
326 006E 47                         jp       hgupi2
327 006F 96E612          hgrst:  ifbit     la0,upic
328 0072 43                         jp       hgupi2
329 0073 B4027A                     jmpl     xreset          ; If so, then go reset the HPC.
330                                                          ; This is part of the outer loop, waiting for
331                                                          ;  the RESET command.
332
333 0076 97F7D2          hgupi2: ld        irpd,#x'F7      ; Clear the UWR detector,
334 0079 7B                         jp       hgupi           ;  and keep looking.  This is an
335                                                          ;  infinite loop until RESET is seen.
336
337 007A 30              hextab: .byte      '0','1','2','3','4','5','6','7'
    007B 31
    007C 32
    007D 33
    007E 34
    007F 35
    0080 36
    0081 37
338 0082 38                         .byte      '8','9','A','B','C','D','E','F'
    0083 39
    0084 41
    0085 42
    0086 43
    0087 44
    0088 45
    0089 46
339
```

TL/DD/9976-25

```
340                                 .form    'Hardware Initialization'
341
342 008A 9708C0          start:  ld        psw.b,#x'08      ; Set one WAIT state.
343
344 008D               srfsh:                       ; Start dynamic RAM refreshing,
345                                                  ;  as quickly as possible.
346 008D B6015208                   sbit     t4out,portpl    ; Trigger first refresh
347                                                          ;  immediately.
348 0091 B601500A                   sbit     t4stp,pwmdl     ; Stop timer T4 to
349                                                          ;  allow loading,
350 0095 83080140AB                 ld       t4.w,#8         ;  then load it.
351 009A B601501A                   rbit     t4stp,pwmdl     ; Start timer T4.
352 009E B601520B                   sbit     t4tfn,portpl    ; Enable pulses out.
353 00A2 83080142AB                 ld       r4.w,#8         ; Load R4.
354
355 00A7               supi:                        ; Set up UPI port.
356 00A7 9718E6                     ld       upic,#x'18      ; 8-Bit UPI Mode
357                                                          ;  enabled.
358
359 00AA 96F50B                     sbit     uwrrdy,bfunh    ; Enable UWRRDY/ out.
360 00AD 96F30B                     sbit     uwrrdy,dirbh
361 00B0 88F0                       ld       A,ibuf          ; Empty IBUF register,
362                                                          ;  in case of false trigger.
363
364 00B2 96F50F                     sbit     urdrdy,bfunh    ; Enable URDRDY/ out.
365 00B5 96F30F                     sbit     urdrdy,dirbh
366                                                          ; Set up UREAD/ interrupt.
367
368 00B8 96D40A                     sbit     i2,ircd         ; Detects rising edges.
369 00BB 97FBD2                     ld       irpd,#x'FB      ; Clear any false interrupt
370                                                          ;  due to mode change.
371
372                                                          ; Set up UWRITE/ interrupt.
373 00BE 96D40B                     sbit     i3,ircd         ; Detects rising edges.
374 00C1 97F7D2                     ld       irpd,#x'F7      ; Clear any false interrupt
375                                                          ;  due to mode change.
376
377 00C4               sram:                        ; Clear all RAM locations.
378                                                  ; Clear Basepage bank:
379 00C4 8D00BE                     ld       BK,#x'0000,#x'00BE     ; Establish loop base and limit.
380 00C7 00              sraml1: clr       A
381 00C8 E1                         xs       A,[B+].w
382 00C9 62                         jp       sraml1
383
384                                                  ; Clear Non-Basepage bank:
385 00CA A701C001FE                 ld       BK,#x'01C0,#x'01FE     ; Establish loop base and limit.
386 00CF 00              sraml2: clr       A
387 00D0 E1                         xs       A,[B+].w
388 00D1 62                         jp       sraml2
389
```

TL/DD/9976-26

```
390 00D2                    sskint:                          ; Set up Stack and remove
391                                                          ;   individual interrupt enables.
392 00D2 B70002C4     R          ld      sp,#stackb+2        ; Move stack to high
393                                                          ;   bank of on-chip RAM.
394 00D6 870000000AB   R          ld      stackb.w,#hangup   ; Safeguard against
395                                                          ;   stack underflow.
396 00DC 9700D0                   ld      enir,#x'00          ; Disable interrupts
397                                                          ;   individually.
398
399 00DF 83080192BB    tminit: ld      t0con,#x'08
400 00E4 87444001AB           ld      tmmode,#x'4440      ; Stop timers T1, T2, T3.
401 00EA 8355018EAB           ld      divby,#x'0055       ; Timers T2 and T3 set to
402                                                          ;   clock externally.
403 00EF 87CCC8019AB           ld      tmmode,#x'CCC8      ; Clear and disable timer
404                                                          ;   T0-T3 interrupts.
405
406 00F5 97FFF1               ld      dirah,#x'FF         ; Initialize Port A upper byte for output.
407 00F8 96E30D               sbit    astts,portbh        ; Enable and de-assert ENASTTS/ signal
408 00FB 96F30D               sbit    astts,dirbh         ;   (enables pushbutton data to Port D).
409 00FE 96E30C               sbit    cdata,portbh        ; Enable and de-assert ENCDATA/ signal.
410 0101 96F30C               sbit    cdata,dirbh         ;   (enables other data to Port D).
411
412 0104 97FFE1       sled:   ld      portah,#x'FF        ; Set up to turn off LED's.
413 0107 96E31E               rbit    ledclk,portbh       ; Start with LEDCLK low,
414 010A 96F30E               sbit    ledclk,dirbh        ;   (enable output),
415 010D 96E30E               sbit    ledclk,portbh       ;   then high,
416 0110 96E31E               rbit    ledclk,portbh       ;   then low again.
417
418 0113                stmrs:                           ; Set up remaining timers.
419                                                          ;   (T1-T3 already stopped
420                                                          ;    and pending bits cleared
421                                                          ;    at tminit above, as
422                                                          ;    part of MICROWIRE init.)
423
424 0113 872FFF0182AB           ld      t1,#12287          ; T1 runs at 10-millisecond real-time interval.
425 0119 872FFF0184AB           ld      r1,#12287
426                                                          ; Timer remains stopped, and interrupt
427                                                          ;   disabled, until INITIALIZE command.
428
429 011F 87444001AB           ld      pwmode,#x'4440     ; Stop timers T4-T7.
430 0125 40                   nop                          ; Wait for valid PND
431 0126 40                   nop                          ;   bits.
432 0127 87CCC801AB           ld      pwmode,#x'CCC8     ; Clear and disable
433                                                          ;    interrupts from all
434                                                          ;    PWM timers.
435
436 012D 87FFFF014AAB           ld      r6,#x'FFFF         ; No modulus for LCD Display Ready timer.
437
438 0133 83CC014CAB           ld      t7,#204            ; Set T7 to underflow at 6 KHz rate
439 0138 83CC014EAB           ld      r7,#204            ;   (= 3 KHz at pin).
```

TL/DD/9976-27

```
440 013D B601531F              rbit    t7tfn,portph   ; Disable beep tone to panel speaker.
441 0141 B601511E              rbit    t7stp,pwmdh    ; Start T7 running.
442
443
444 0145              slcd:                           ; Set up LCD display.
445                                                   ; Requires use of timer T6, so
446                                                   ;   appears after timer initialization.
447
448                                                   ; First, set up LCD contrast.
449 0145 970A12       R         ld     lcvs,#x'0A     ; Initialize memory image of LCD Voltage
450                                                   ;   latch, containing RS (PAUX0) bit also.
451 0148 8C12E1       R         ld     portah,lcvs    ; Arbitrary initial contrast level of 5,
452                                                   ;   and RS/ (PAUX0/) is high (="command").
453 014B 96E319                 rbit   lcvclk,portbh  ; Start with LCVCLK low,
454 014E 96F309                 sbit   lcvclk,dirbh   ;   (enable output)
455 0151 96E309                 sbit   lcvclk,portbh  ;   then high,
456 0154 96E319                 rbit   lcvclk,portbh  ;   then low to get it into LCV latch.
457
458                                                   ; Initialize PNLCLK (Panel "E" signal).
459 0157 96E20F                 sbit   pnlclk,portbl  ; Start with PNLCLK high
460 015A 96F20F                 sbit   pnlclk,dirbl   ;   (enable output).
461
462                                                   ; Wait for worst-case command
463                                                   ;   execution time (4.9 ms, twice), in case
464                                                   ;   a panel command was triggered while
465                                                   ;   PNLCLK was floating.
466 015D B601510B               sbit   t6ack,pwmdh    ; Clear T6 PND bit.
467 0161 8732C80148AB           ld     t6,#13000      ; Set T6 to twice 4.9 milliseconds.
468 0167 B601511A               rbit   t6stp,pwmdh    ; Start timer T6.
469 016B B6015111     lcdlp1:   ifbit  t6pnd,pwmdh    ; Wait for T6 PND bit
470                                                   ;   to be set.
471 016F 41                     jp     lcdgo1
472 0170 65                     jp     lcdlp1
473 0171 B601510A     lcdgo1:   sbit   t6stp,pwmdh    ; Stop timer T6.
474 0175 B601510B               sbit   t6ack,pwmdh    ; Clear T6 PND bit.
475
476                                                   ; Reset Panel controller (per Hitachi HD44780
477                                                   ;   User's Manual).
478
479                                                   ; (Panel RS signal was set
480                                                   ;   in LCD Contrast initialization above,
481                                                   ;   so no change needed here to
482                                                   ;   flag these as commands.)
483
484 0179 9038                   ld     A,#x'38        ; Send "8-Bit Mode, 2 Lines" command:  one;
485 017B 2E           R         jsrl   wrpnl
486 017C 9038                   ld     A,#x'38        ; two;
487 017E 2E           R         jsrl   wrpnl
488 017F 9038                   ld     A,#x'38        ; three;
489 0181 2E           R         jsrl   wrpnl
```

TL/DD/9976-28

```
490 0182 9038                   ld     A,#x'38        ; four times.
491 0184 2E           R         jsrl   wrpnl
492 0185 9008                   ld     A,#x'08        ; Disable display.
493 0187 2E           R         jsrl   wrpnl
494 0188 9001                   ld     A,#x'01        ; Clear display RAM.
495 018A 2E           R         jsrl   wrpnl
496
497                                                   ; Initial default mode settings.
498
499 018B 9006                   ld     A,#x'06        ; Set mode to move cursor to the right, no
500 018D 2E           R         jsrl   wrpnl          ;   automatic shifting of display.
501 018E 900E                   ld     A,#x'0E        ; Enable display:  non-blinking cursor mode.
502 0190 2E           R         jsrl   wrpnl
503
504
505                    ;         CONTINUES TO MAIN PROGRAM INITIALIZATION
```

TL/DD/9976-29

```
506                                .form   'Main Program Initialization'
507
508
509 0191              minit:
510                                        ; Once-only initializations.
511
512 0191 978010       R      ld     curcmd,#x'80    ; Current Command:  top bit set means "none".
513 0194 B7000604      R      ld     cpuad,#cpubuf   ; Set CPU command index to beginning of buffer.
514 0198 970811        R      ld     numexp,#8       ; Arbitrary starting value.
515
516                                        ; Arbitrary set of initialization values for variables,
517                                        ;  in effect until receipt of the first INITIALIZE
518                                        ;  command.
519
520 019B B7000002      R      ld     alert,#0        ; No events pending.
521
522 019F              runsys:                  ; Enable interrupts, start timers and go to main loop.
523
524 019F 96D00D               sbit   tmrs,enir       ; Enable timer interrupts.  (Done here
525                                        ;   to allow certain commands without an
526                                        ;   INITIALIZE command first.)
527 01A2 96D00B               sbit   i3,enir         ; Enable CPU Command interrupt.
528 01A5 96D008               sbit   gie,enir        ; Enable interrupt system.
529
530
```

```
531                                .form   'Main Scan Loop'
532
533                         ;      Declarations
534
535 0011              vrtc    =      x'11     ; Real-Time Clock vector number.
536 0017              vlcdak  =      x'17     ; Acknowledge finished writing to LCD panel.
537 0018              vbutton =      x'18     ; Pushbutton status change: a button pressed or
538                                           ;  released.
539 001D              vdiag   =      x'1D     ; Diagnostic Interrupt.
540
541
542                                        ; Error Vectors for unimplemented or
543                                        ;   unexpected interrupts.
544
545                         ;      level   0  is Reset, provided by assembler.
546 FFFC 0000      R      .ipt   1,hangup        ; NMI:  never expected.
547 FFFA 0000      R      .ipt   2,hangup        ; UPI READ READY:  never expected.
548 FFF6 0000      R      .ipt   4,hangup        ; I4 Interrupt Vector:  never expected.
549 FFF2 0000      R      .ipt   6,hangup        ; UART Interrupt Vector:  never expected.
550 FFF0 0000      R      .ipt   7,hangup        ; EI Interrupt Vector:  never expected.
551
552 01A8              mainlp:
553
554
555 01A8 820002FC      R  chkalt: ifeq   alert.w,#x'00   ; Check for alert conditions.
556 01AC 64                 jp     chkalt          ; If none, keep looping.
557
558 01AD 960211       R      ifbit  artc,alert.b    ; Check for RTC interrupt request.
559 01B0 3010                jsrl   sndrtc          ; If so, then send Real-Time Clock interrupt.
560
561 01B2 960213       R      ifbit  alcdak,alert.b  ; Check for LCD Panel write done.
562 01B5 3013                jsrl   sndlak          ; If so, then send LCD Acknowledge interrupt.
563
564 01B7 960210       R      ifbit  abutton,alert.b ; Check for a pushbutton change.
565 01BA 3016                jsrl   sndbtn          ; If so, then report the change to the CPU.
566
567 01BC 960212       R      ifbit  adiag,alert.b   ; Check for Diagnostic Interrupt.
568 01BF 3023                jsrl   sndiag          ; If so, then send interrupt and data.
569
570 01C1 79                  jmpl   chkalt          ; No "responses" defined yet;  just close loop.
571
```

```
572                                .form   'Main:  Send Real-Time Clock Interrupt'
573
574                         ; No data transfer;  just trigger interrupt and continue.
575
576 01C2              sndrtc:
577 01C2 960219       R      rbit   artc,alert.b    ; Clear ALERT bit.
578 01C5 2F           R      jsrl   rdwait          ; Check that UPI interface is ready.
579                                        ; If not, loop until it is.
580
581 01C6 9711E0              ld     obuf,#vrtc      ; Load Real-Time Clock vector into OBUF for CPU.
582 01C9 3C                  ret                    ; Return to main loop.
583
```

```
     584                                   .form    'Main:  Send LCD Write Acknowledge Interrupt'
     585
     586                          ; No data transfer;  just trigger interrupt and continue.
     587
     588 01CA                sndlak:
     589 01CA 96021B     R         rbit    alcdak,alert.b  ; Clear ALERT bit.
     590 01CD 2F         R         jsrl    rdwait          ; Check that UPI interface is ready.
     591                                                   ; If not, loop until it is.
     592
     593 01CE 9717E0               ld      obuf,#vlcdak    ; Load LCD-Acknowledge vector into OBUF for CPU.
     594 01D1 3C                   ret                     ; Return to main loop.
     595
```

                                                                                          TL/DD/9976-33

```
     596                                   .form    'Main:  Send Pushbutton Status to CPU'
     597
     598 01D2                sndbtn:
     599 01D2 2F         R         jsrl    rdwait          ; Check that UPI interface is ready.
     600                                                   ; If not, loop until it is.
     601
     602 01D3 9718E0               ld      obuf,#vbutton   ; Load BUTTON-DATA vector into OBUF for CPU.
     603
     604 01D6 2F         R         jsrl    rdwait          ; Check that UPI interface is ready.
     605                                                   ; If not, loop until it is.
     606
     607 01D7 960018               rbit    gie,enir        ; *** Begin Indivisible Sequence ***
     608 01DA 8C18E0     R         ld      obuf,swlsnt     ; Load Pushbutton Data Byte into OBUF for CPU.
     609 01DD 960218     R         rbit    abutton,alert.b ; Clear ALERT bit.
     610 01E0 960008               sbit    gie,enir        ; *** End Indivisible Sequence ***
     611 01E3 3C                   ret                     ; Return to main loop.
     612
```

                                                                                          TL/DD/9976-34

```
613                              .form    'Main:  Send Diagnostic Interrupt to CPU'
614
615 01E4                 sndiag:
616 01E4 2F          R          jsrl    rdwait      ; Wait for UPI interface ready.
617 01E5 971DE0                 ld      obuf,#vdiag  ; Load vector into OBUF for CPU.
618 01E8 2F          R          jsrl    rdwait      ; Wait for UPI interface ready.
619 01E9 96D018                 rbit    gie,enir    ; *** Begin Indivisible Sequence ***
620 01EC 8C1DE0       R          ld      obuf,dsevc   ; Transfer Severity Code.
621 01EF 97001D       R          ld      dsevc,#0    ; Clear it.
622 01F2 881E         R          ld      A,derrc     ; Get Error Code.
623 01F4 97001E       R          ld      derrc,#0    ; Clear it.
624 01F7 96D21A       R          rbit    adiag,alert.b  ; Clear ALERT bit.
625 01FA 96D008                 sbit    gie,enir    ; *** End Indivisible Sequence ***
626 01FD 2F          R          jsrl    rdwait      ; Wait for UPI interface ready.
627 01FE 8BE0                   st      A,obuf      ; Transfer Error Code.
628 0200 2F          R          jsrl    rdwait      ; Wait for UPI interface ready.
629                                                 ; Remaining bytes will have meaning only for
630                                                 ;   command errors.
631 0201 8C1FE0       R          ld      obuf,dbyte   ; Transfer Byte Received.
632 0204 2F          R          jsrl    rdwait      ; Wait for UPI interface ready.
633 0205 8C20E0       R          ld      obuf,dccmd   ; Transfer Current Command.
634 0208 2F          R          jsrl    rdwait      ; Wait for UPI interface ready.
635 0209 8C21E0       R          ld      obuf,dqual   ; Transfer Command Count.
636 020C 3C                     ret                 ; Return to main program loop.
637
```

                                                                                    TL/DD/9976-35

```
638                              .form    'UPI (I3) Interrupt:  Data from CPU'
639
640 FFF8 0D02        R          .ipt    3,upiwr     ; Declare upiwr as vector for Interrupt 3.
641
642 020D                 upiwr:                      ; Write Strobe received from CPU.
643 020D AFC8                   push    A           ; Save Context
644 020F AFC0                   push    psw
645
646 0211 8CE600       R          ld      upicsv.b,upic  ; Save UPIC register image for LA0 bit test.
647
648 0214 961017       R          ifbit   cmdemp,curcmd  ; If expecting first byte of a command,
649 0217 94CC                    jmpl    firstc      ;   then go process it as such.
650
651 0219 88F0                   ld      A,ibuf      ; If not, input it for entry into cpubuf.
652
653 021B 9CA5                   ifeq    A,#x'A5     ; Check for RESET command.
654 021D 46                     jp      lcrst
655 021E 960012       R          ifbit   la0,upicsv.b  ; Check for command argument written to proper
656                                                 ;   address.
657 0221 48                     jp      lcord       ; If so, go process as a normal argument.
658 0222 3622                   jsrl    hangup      ; If not, process as a FATAL error, generating
659                                                 ;   !DIAG interrupt.
660
661 0224 96E612                 lcrst:  ifbit   la0,upic    ; Continue checking for a RESET command.
662 0227 42                     jp      lcord
663 0228 94C6                   jmpl    xreset      ; If so, go reset the HPC.
664
665 022A AD048E       R  lcord:  x       A,[cpuad].b  ; If not, place it in next available cpubuf
666                                                 ;   entry.
667 022D A904         R          inc     cpuad
668 022F 8A11         R          decsz   numexp
669 0231 B4010F                 jmpl    upwret      ; If not final byte of command, then return.
670
671 0234 8810         R  lastc:  ld      A,curcmd     ; Else, process current command.
672 0236 96C816                 ifbit   getcnt,A.b   ; Check if extended collection is being made.
673 0239 47                     jp      lastc1      ; If not, then:
674 023A 961000F      R          sbit    cmdemp,curcmd  ;  Set command slot available again.
675 023D B7000604     R          ld      cpuad,#cpubuf  ;  Reset CPU buffer pointer to beginning.
676
677 0241 991F                 lastc1: and     A,#x'1F     ; Mask off flag bits.
678 0243 E7                     shl     A           ; Scale by two, and then
     0244 40
679 0245                        .odd
680 0245 EC                     jidw                ;  jump based on command value:
681
682 0246 0A00                 lastab: .ptw    lcinit      ; 0 = INITIALIZE command.
683 0248 2C00                        .ptw    lcslcv      ; 1 = SET-CONTRAST command.
684 024A 4200                        .ptw    lcslcd      ; 2 = SEND-LCD command.
685 024C 8C00                        .ptw    lcsled      ; 3 = SEND-LED command.
686 024E F300                        .ptw    illc        ; (BEEP command has only one byte.  Error.)
```

                                                                                    TL/DD/9976-36

```
687
688                                                ; Process INITIALIZE Command.
689
690 0250 97011C        R   lcinit: ld     rtevs,#x'01     ; Enable only Real-Time Clock interrupts, but
691 0253 8200060C      R           ifeq   cpubuf.b,#0     ;  disable them again if
692 0257 961C18        R           rbit   rtcenb,rtevs    ;  the command argument is zero.
693 025A 8C061A        R           ld     rtcivl,cpubuf.b ; Put argument into Real-Time
694                                                        ;  Clock interval.
695 025D 8C061B        R           ld     rtccnt,cpubuf.b ; Put argument into Real-Time
696                                                        ;  Clock count.
697 0260 B601900C                  sbit   t1tie,tmmdl     ; Enable Timer T1 interrupt, if not already
698                                                        ;  enabled.
699 0264 B601901E                  rbit   t1stp,tmmdl     ; Start timer, if not already running.
700
701 0268 B7000002      R           ld     alert.w,#0      ; Set no events pending.
702
703 026C 970017        R           ld     swlast,#0       ; Set up initial switch values.
704 026F 970018        R           ld     swlsnt,#0       ; (Both current and last sent)
705
706 0272 94CF                      jmpl   upwret          ; Return.
707
708
709                                                ; Process SET-CONTRAST Command.
710
711 0274 8806          R   lcslcv: ld     A,cpubuf.b      ; Load LCD Voltage latch (Contrast) from byte
712                                                        ;  supplied by CPU.
713 0276 01                        comp   A               ; (3-bit value is in complemented form.)
714 0277 9907                      and    A,#x'07         ; Use only lower three bits.
715 0279 82F812D9      R           and    lcvs,#x'F8      ; Clear field in memory image.
716 027D 80C812DA      R           or     lcvs,A.b        ; Merge new field into image.
717 0281 8C12E1        R           ld     portah,lcvs     ; Place on Port A (input to latch).
718 0284 96E309                    sbit   lcvclk,portbh   ; Clock latch.
719 0287 96E319                    rbit   lcvclk,portbh
720 028A 94B7                      jmpl   upwret
721
722
723                                                ; Process SEND-LCD Command.
724
725 028C 961016        R   lcslcd: ifbit  getcnt,curcmd      ; Check for first or second collection
726 028F 9435                      jmpl   lcslc1             ;  phase.
727
728 0291                  lcslc2:                            ; Second phase:  begins execution of the LCD
729                                                          ;  command.
730 0291 A1060038AB    R           ld     lcdbuf.w,cpubuf.w    ; Copy CPU buffer to LCD string buffer.
731 0296 A108003AAB    R           ld     lcdbuf+2.w,cpubuf+2.w
732 029B A10A003CAB    R           ld     lcdbuf+4.w,cpubuf+4.w
733 02A0 A10C003EAB    R           ld     lcdbuf+6.w,cpubuf+6.w
734 02A5 8C1416        R           ld     lcdsct,lcdnum      ; Move number of characters to string
735                                                          ;  count byte
736 02A8 8916          R           inc    lcdsct             ;  (incremented by one because of
```

TL/DD/9976-37

```
737                                                        ;   extra interrupt occurring after
738                                                        ;   last character has been sent).
739 02AA B700380E      R           ld     lcdsix,#lcdbuf  ; Set string pointer to first byte.
740 02AE 8C1315        R           ld     lcdsfg,lcdfgs   ; Move flag bits to string location.
741
742 02B1 87FFFF014AAB            ld     r6,#x'FFFF        ; Set up R6 and T6 to trigger string
743 02B7 83000148AB             ld     t6,#0             ;  transfer.
744 02BC B6015108                sbit   t6tie,pwmdh       ; Enable timer T6 interrupt.
745 02C0 B601511A                rbit   t6stp,pwmdh       ; Start timer to trigger (immediate)
746                                                        ;  interrupt from timer T6.
747 02C4 947D                    jmpl   upwret
748
749 02C6                  lcslc1:                          ; First phase:  Prepare to collect up to 8
750                                                        ;  more bytes of command.
751 02C6 8C0613        R           ld     lcdfgs,cpubuf.b ; Get flag bits supplied by CPU.
752 02C9 8C0714        R           ld     lcdnum,cpubuf+1.b ; Get character count from CPU.
753
754 02CC 8C1411        R           ld     numexp,lcdnum.b ; Request another collection of
755                                                        ;  data from the CPU (the string of
756                                                        ;  data for the panel).
757 02CF B7000604      R           ld     cpuad,#cpubuf   ; Reset CPU collection pointer to start
758                                                        ;  of command buffer.
759 02D3 96101E        R           rbit   getcnt,curcmd   ; Declare that it will be the final
760                                                        ;  collection.
761 02D6 946B                      jmpl   upwret
762
763
764                                                ; Process SEND-LED Command.
765
766 02D8 8806          R   lcsled: ld     A,cpubuf.b      ; Load LED latch from byte supplied by CPU.
767 02DA 01                        comp   A               ; (Data goes to LED's in complemented form.)
768 02DB 8BE1                      st     A,portah        ; Place new value on Port A (input to latch).
769 02DD 96E30E                    sbit   ledclk,portbh   ; Clock latch.
770 02E0 96E31E                    rbit   ledclk,portbh
771 02E3 945E                      jmpl   upwret
772
773
```

TL/DD/9976-38

```
774                                  .form    'Processing of First Byte of Command (Code)'
775
776                                  ;  One-byte commands are processed in this section.
777                                  ;  Longer commands are scheduled for collection of
778                                  ;    remaining bytes, and are processed in routines
779                                  ;    above.
780
781 02E5 88F0         firstc: ld     A,ibuf              ; Get command from UPI port.
782 02E7 960012     R         ifbit  la0,upicsv.b        ; Check for out-of-sequence condition
783                                                       ;  (argument instead of command).
784 02EA 36EA                 jsrl   hangup              ; If so, process as a FATAL error (previous
785                                                       ;  command was too short).
786
787                                          ; Processing of RESET command.
788
789 02EC 9CA5                 ifeq   A,#x'A5             ; Check for RESET command.
790 02EE 41                   jp     xreset
791 02EF 59                   jp     fcord
792
793                                                       ; This code is entered whenever a RESET
794                                                       ;  command is received.
795 02F0         xreset:
796 02F0 971DE0               ld     obuf,#vdiag         ; Present dummy value for CPU,
797                                                       ;  (in case a value was already in OBUF),
798 02F3 2F        R         jsrl   rdwait              ;  and wait for it to be read by CPU.
799 02F4 9000                 ld     A,#0                ; Initialize registers.
800 02F6 88E6                 st     A,upic
801 02F8 ABF0                 st     A,ibuf.w            ; (Actually all of DIRA.)
802 02FA ABF2                 st     A,dirb
803 02FC ABF4                 st     A,bfun
804 02FE 88D4                 st     A,ircd
805 0300 B60152AB             st     A,portp
806 0304 ABC4                 st     A,sp                ; Then, through RESET vector,
807 0306 ABC0                 st     A,psw
808 0308 3C                   ret                        ;  jump to start of program.
809
810                                          ; Here, process an ordinary command (not RESET).
811
812 0309         fcord:
813 0309 991F                 and    A,#x'1F             ; Use only least-significant 5 bits.
814 030B 9D11                 ifgt   A,#x'11             ; Check for command out of range.
815 030D 9432                 jmpl   illc
816 030F 8B10      R         st     A,curcmd            ; Save as current command.
817
818 0311 E7                   shl    A                   ; Scale by two, and then
    0312 40
819 0313                      .odd
820 0313 EC                   jidw                       ;  jump based on command value:
821
822 0314 0A00         firstab: .ptw   fcinit             ; 0 = INITIALIZE command.
```

TL/DD/9976-39

```
823 0316 0D00                 .ptw   fcslcv             ; 1 = SET-CONTRAST command.
824 0318 0F00                 .ptw   fcslcd             ; 2 = SEND-LCD command.
825 031A 1400                 .ptw   fcsled             ; 3 = SEND-LED command.
826 031C 1600                 .ptw   fcbeep             ; 4 = BEEP command.
827
828 031E 970111    R  fcinit: ld     numexp,#1           ; First byte of INITIALIZE command.
829                                                       ;  Expects 1 more byte (RTC interval).
830 0321 9420                 jmpl   upwret              ; Return.
831
832                                                       ; First byte of SET-CONTRAST command.
833 0323 970111    R  fcslcv: ld     numexp,#1           ; Set up to expect one more byte.
834 0326 5C                   jmpl   upwret
835
836                                                       ; First byte of SEND-LCD command.
837 0327 970211    R  fcslcd: ld     numexp,#2           ; Set up to expect one more byte.
838 032A 96100E    R         sbit   getcnt,curcmd       ; Note extended collection mode in Current
839                                                       ;  Command byte.
840 032D 55                   jmpl   upwret
841
842                                                       ; First byte of SEND-LED command.
843 032E 970111    R  fcsled: ld     numexp,#1           ; Send to LED's: Set up to expect one more byte.
844 0331 51                   jmpl   upwret
845
846                                                       ; Process one-byte BEEP command.
847 0332 96100F    R  fcbeep: sbit   cmdemp,curcmd       ; No arguments; set CURCMD byte empty.
848 0335 B60153 0F            sbit   t7tfn,portph        ; Enable beep tone to panel speaker.
849 0339 B6019008             sbit   t0tie,tmmdl         ; Enable Timer T0 interrupt.
850 033D 971319    R         ld     beepct,#19          ; Initialize duration count (approximately
851                                                       ;  1 second, in units of Timer T0 overflows).
852 0340 42                   jmpl   upwret
853
854
855 0341 3741         illc:   jsrl   hangup              ; Process illegal command codes.
856
857                                                       ; Return from UPI Write interrupt.
858 0343         upwret:                                  ; Restore Context
859 0343 3FC0                 pop    psw
860 0345 3FC8                 pop    A
861 0347 3E                   reti
862
```

TL/DD/9976-40

```
863                              .form   'Timer Interrupt Handler'
864
865 FFF4 4803       R            .ipt    5,tmrint      ; Declare entry point for Timer Interrupt.
866
867 0348 AFC8              tmrint: push   A             ; Save context.
868 034A AFCC                      push   B             ;
869 034C AFC0                      push   psw           ;
870
871 034E B6019015         t1poll: ifbit  t1pnd,tmmdl   ; Poll for Timer T1 interrupt (Real-Time Clock).
872 0352 54                        jmpl   t1int         ; If set, go service it.
873
874 0353 B6015111         t6poll: ifbit  t6pnd,pwmdh   ; Poll for Timer T6 interrupt (LCD Panel Timing
875 0357 944C                      jmpl   t6int         ;   Interrupt).
876
877 0359 B6019011         t0poll: ifbit  t0pnd,tmmdl   ; Poll for Timer T0 interrupt (Beep Duration).
878 035D 41                        jp     t0pdg         ; If set, check the Enable bit;  T0 is not
879 035E 46                        jp     t0notp        ;   always enabled to interrupt, but it runs
880                                                      ;   continuously.
881
882 035F B6019010         t0pdg:  ifbit  t0tie,tmmdl   ; If enable is also set, then go service T0.
883 0363 9488                      jmpl   t0int
884 0365             t0notp:                            ; (This label is deliberately here.)
885
886 0365 3765             noint:  jsrl   hangup        ; Error:  no legal timer interrupt pending.
887
888
```

```
889                              .form   'Timer T1 Interrupt Service Routine'
890
891 0367 B601900F         t1int:  sbit   t1ack,tmmdl   ; Acknowledge T1 interrupt.
892 036B 961C10       R           ifbit  rtcenb,rtevs  ; Check if RTC interrupts are enabled.
893 036E 41                       jp     t1int1
894 036F 57                       jmpl   kbdchk        ; If not, then go check other events.
895 0370 8A1B         R   t1int1: decsz  rtccnt        ; Decrement interval value.
896 0372 54                       jmpl   kbdchk        ; If interval has not elapsed, then go check
897                                                     ;   for other events.
898 0373 8C1A1B       R           ld     rtccnt,rtcivl ; Reload counter value for next interval.
899 0376 960211       R           ifbit  artc,alert.b  ; Check if CPU has received previous interrupt
900 0379 44                       jp     t1rerr        ;   request;  report error if not.
901 037A 960209       R           sbit   artc,alert.b  ; Set Real-Time Interrupt request to main
902 037D 49                       jp     kbdchk        ;   program.
903 037E 961D08       R   t1rerr: sbit   0,dsevc       ; Signal NOTE severity.
904 0381 961E0F       R           sbit   7,derrc       ; Signal multiple-RTC error.
905 0384 96020A       R           sbit   adiag,alert.b ; Request !DIAG interrupt from main program.
906
907 0387             kbdchk:                           ; Check keyboard switches.
908 0387 96E31D               rbit   astts,portbh  ; Enable pushbutton data to Port D.
909 038A 86010488             ld     A,portd       ; Sample pushbutton switches.
910 038E 96E30D               sbit   astts,portbh  ; Disable pushbutton data to Port D.
911 0391 9BFF                 xor    A,#x'FF       ; Complement low-order 8 bits of A.
912 0393 8E17         R       x      A,swlast      ; Exchange with last sample.
913 0395 9617DC       R       ifeq   A,swlast      ; Check if the data is stable (same as last
914                                                 ;   sample).
915 0398 41                   jp     kbint1
916 0399 49                   jmpl   tmochk        ; If not, go check other events (if any).
917
918 039A 9618DC       R   kbint1: ifeq   A,swlsnt      ; Check if the data differs from the last
919                                                 ;   pattern sent to the CPU.
920 039D 45                   jmpl   tmochk        ; If not, go check other events (if any).
921
922 039E 8B18         R       st     A,swlsnt      ; Place new pattern in "last sent" location.
923 03A0 960208       R       sbit   abutton,alert.b ; Request "BUTTON-DATA" interrupt to CPU.
924
925
926 03A3             tmochk:
927                          ; *** Insert any other RTC events here. ***
928
929 03A3 9459                jmpl   tmrret        ; Return from Timer T1 interrupt.
930
931
```

```
932                                 .form    'Timer T6 Interrupt Service Routine'
933
934                                         ; Timer T6 interrupt routine: sends characters from
935                                         ;   LCD String Buffer to the panel.
936  03A5 B601510A      t6int:  sbit    t6stp,pwmdh     ; Stop timer T6.
937  03A9 B601510B              sbit    t6ack,pwmdh     ; Acknowledge T6 interrupt.
938
939  03AD 8A16       R          decsz   lcdsct          ; Decrement LCD character count.
940  03AF 45                    jmpl    t6nxtc          ; If not done, go send another character.
941
942  03B0 96020B     R          sbit    alcdak,alert.b  ; If done, request main program to send LCD
943                                                     ;   Acknowledge interrupt to CPU.
944  03B3 9449                  jmpl    tmrret
945
946  03B5 8815       R  t6nxtc: ld      A,lcdsfg        ; Get flags byte (for panel RS signal).
947  03B7 C7                    shr     A               ; Shift right, LSB into carry.
948  03B8 8815       R          st      A,lcdsfg        ; Store shifted value back.
949  03BA 96120B     R          sbit    pnlrs,lcvs      ; Determine proper state for RS signal from
950  03BD 07                    ifc                     ;   current character's flag (= flag inverted).
951  03BE 96121B     R          rbit    pnlrs,lcvs
952  03C1 8C12E1     R          ld      portah,lcvs     ; Send new RS value to LCD Voltage (LCV) latch.
953  03C4 96E309                sbit    lcvclk,portbh   ; Clock the latch.  RS signal is now valid.
954  03C7 96E319                rbit    lcvclk,portbh
955
956  03CA AD0E88     R          ld      A,[lcdsix].b    ; Get next LCD character from string buffer.
957  03CD A90E       R          inc     lcdsix          ; Increment character pointer.
958  03CF 01                    comp    A               ; Complement character, then
959  03D0 8BE1                  st      A,portah        ;   place it on Port A for LCD display.
960  03D2 96E21F                rbit    pnlclk,portbl   ; Clock it into panel.
961  03D5 96E20F                sbit    pnlclk,portbl
962  03D8 01                    comp    A               ; Restore A to uncomplemented form for
963                                                     ;   test performed below.
964
965  03D9 83940148AB            ld      t6,#148         ; Set up normal delay time in timer T6
966                                                     ;   (120 microseconds).
967  03DE 9D03                  ifgt    A,#x'03         ; Check whether the longer delay
968  03E0 47                    jp      t6nxt2          ;   (4.9 milliseconds) is necessary.
969                                                     ;   This happens if RS=0 and the byte sent to
970  03E1 06                    ifnc                    ;   the panel is a value of hex 03 or less.
971  03E2 8717860148AB          ld      t6,#6022        ; If so, change timer to 4.9 milliseconds.
972
973  03E8 B601511A     t6nxt2: rbit    t6stp,pwmdh     ; Start Timer T6 to time out the character.
974  03EC 51                    jmpl    tmrret          ; Return from the interrupt.
975
976
```

```
977                                 .form    'Timer T0 Interrupt Service Routine'
978
979  03ED               t0int:               ; Count duration of beep tone.  Restore beep signal
980                                           ;   to zero and re-enable switch sampling interrupt
981                                           ;   when done.
982  03ED B601900B              sbit    t0ack,tmmdl     ; Acknowledge interrupt from Timer T0.
983  03F1 8A19       R          decsz   beepct          ; Check whether beep time has finished.
984  03F3 4A                    jmpl    tmrret          ;  No:  return from interrupt.
985  03F4 B6019018              rbit    t0tie,tmmdl     ;  Yes:  disable Timer T0 interrupts and
986                                                     ;         continue.
987  03F8 830F0153D9            and     portph,#x'0F    ; Disable speaker output.
988  03FD 40                    jmpl    tmrret          ; Return from interrupt.
989
990                                           ; Common return for timer interrupt service routines.
991  03FE 3FC0       tmrret: pop     psw             ; Restore context.
992  0400 3FCC                  pop     B
993  0402 3FC8                  pop     A
994  0404 3E                    reti
995
996
```

```
997                                 .form    'Subroutine to Wait for OBUF Empty'
998
999                         ;     RDWAIT subroutine:  waits until the CPU has read a byte from the
1000                        ;                         UPI interface.
1001
1002 0405 96E611     rdwait: ifbit   rdrdy,upic      ; Check to see if OBUF register is full.
1003 0408 3C                    ret
1004 0409 64                    jp      rdwait
1005
1006
```

```
1007                                    .form   'Write to Panel Subroutine'
1008
1009                                            ; Write Panel subroutine.
1010                                            ; Used only at initialization or to report a
1011                                            ; fatal protocol error, since it performs
1012                                            ; the timing delay using timer T6 without interrupts.
1013                                            ;   (Panel RS signal must be set up previously in the
1014                                            ;    LCV latch by the calling routine.)
1015
1016  040A 01               wrpnl:  comp    A               ; Complement value for bus.
1017  040B 8BE1                     st      A,portah        ; Put value on panel bus.
1018  040D 96E21F                   rbit    pnlclk,portbl   ; Set Panel Clock low,
1019  0410 96E20F                   sbit    pnlclk,portbl   ;   then high again;
1020                                                        ;   pulse width approx.
1021                                                        ;   1.2 microsec.
1022
1023                                            ; Wait for another
1024                                            ; 4.9 milliseconds (twice).
1025  0413 8732C80148AB            ld      t6,#13000       ; Twice 4.9 milliseconds.
1026  0419 B601511A                 rbit    t6stp,pwmdh     ; Start timer T6.
1027  041D B6015111       wrplp:  ifbit   t6pnd,pwmdh     ; Wait for PND to be set.
1028  0421 41                       jp      wrpgo
1029  0422 65                       jp      wrplp
1030  0423 B601510A       wrpgo:  sbit    t6stp,pwmdh     ; Stop timer T6.
1031  0427 B601510B                sbit    t6ack,pwmdh     ; Clear T6 PND bit.
1032  042B 3C                       ret                     ; Return from subroutine.
1033
1034                         ; END OF PROGRAM:  RESET VECTOR SET TO LABEL "start".
1035
1036  042C                          .end    start
```

TL/DD/9976-46

```
abutton   0000   Abs   Null
adiag     0002   Abs   Null
ah        00C9   Abs   Byte
al        00C8   Abs   Byte
alcdak    0003   Abs   Null
alert     0002   Rel   Word   BASE
alerth    0003   Rel   Byte   BASE
artc      0001   Abs   Null
astts     0005   Abs   Null
avail     0020   Rel   Word   RAM16
b2stp     0007   Abs   Null
b8or16    0004   Abs   Null
b8or9     0004   Abs   Null
beepct    0019   Rel   Byte   BASE
bfun      00F4   Abs   Word
bfunh     00F5   Abs   Byte
bfunl     00F4   Abs   Byte
bh        00CD   Abs   Byte
bl        00CC   Abs   Byte
cdata     0004   Abs   Null
chkalt    01A8   Rel   Null   ROM16
cmdemp    0007   Abs   Null
cpuad     0004   Rel   Word   BASE
cpubuf    0006   Rel   Word   BASE
curcmd    0010   Rel   Byte   BASE
dbyte     001F   Rel   Byte   BASE
dccmd     0020   Rel   Byte   BASE
derrc     001E   Rel   Byte   BASE
dirah     00F1   Abs   Byte
dirb      00F2   Abs   Word
dirbh     00F3   Abs   Byte
dirbl     00F2   Abs   Byte
divby     018E   Abs   Word
divbyh    018F   Abs   Byte
divbyl    018E   Abs   Byte
doeerr    0007   Abs   Null
dqual     0021   Rel   Byte   BASE
dsevc     001D   Rel   Byte   BASE
dummy     0000   Rel   Word   BASE
ei        0007   Abs   Null
eiack     0002   Abs   Null
eicon     015C   Abs   Byte
eimode    0001   Abs   Null
eipol     0000   Abs   Null
enir      00D0   Abs   Byte
enu       0120   Abs   Byte
enui      0122   Abs   Byte
enur      0128   Abs   Byte
eri       0001   Abs   Null
eti       0000   Abs   Null
```

TL/DD/9976-47

```
fcbeep    0332   Rel   Null   ROM16
fcinit    031E   Rel   Null   ROM16
fcord     0309   Rel   Null   ROM16
fcslcd    0327   Rel   Null   ROM16
fcslcv    0323   Rel   Null   ROM16
fcsled    032E   Rel   Null   ROM16
firstab   0314   Rel   Null   ROM16
firstc    02E5   Rel   Null   ROM16
frmerr    0006   Abs   Null
getcnt    0006   Abs   Null
gie       0000   Abs   Null
hangup    0000   Rel   Null   ROM16
hextab    007A   Rel   Byte   ROM16
hgrst     006F   Rel   Null   ROM16
hgupi     005E   Rel   Null   ROM16
hgupi1    0069   Rel   Null   ROM16
hgupi2    0076   Rel   Null   ROM16
i2        0002   Abs   Null
i3        0003   Abs   Null
i4        0004   Abs   Null
ibuf      00F0   Abs   Byte
illc      0341   Rel   Null   ROM16
ircd      00D4   Abs   Byte
irpd      00D2   Abs   Byte
kbdchk    0387   Rel   Null   ROM16
kbint1    039A   Rel   Null   ROM16
la0       0002   Abs   Null
lastab    0246   Rel   Null   ROM16
lastc     0234   Rel   Null   ROM16
lastc1    0241   Rel   Null   ROM16
lcdbuf    0038   Rel   Word   RAM16
lcdfgs    0013   Rel   Byte   BASE
lcdgo1    0171   Rel   Null   ROM16
lcdlp1    016B   Rel   Null   ROM16
lcdnum    0014   Rel   Byte   BASE
lcdsct    0016   Rel   Byte   BASE
lcdsfg    0015   Rel   Byte   BASE
lcdsix    000E   Rel   Word   BASE
lcinit    0250   Rel   Null   ROM16
lcord     022A   Rel   Null   ROM16
lcrst     0224   Rel   Null   ROM16
lcslc1    02C6   Rel   Null   ROM16
lcslc2    0291   Rel   Null   ROM16
lcslcd    028C   Rel   Null   ROM16
lcslcv    0274   Rel   Null   ROM16
lcsled    02D8   Rel   Null   ROM16
lcvclk    0001   Abs   Null
lcvs      0012   Rel   Byte   BASE
ledclk    0006   Abs   Null
mainlp    01A8   Rel   Null   ROM16
```

TL/DD/9976-48

```
minit     0191   Rel   Null   ROM16
noint     0365   Rel   Null   ROM16
numexp    0011   Rel   Byte   BASE
obuf      00E0   Abs   Byte
pnlclk    0007   Abs   Null
pnlrs     0003   Abs   Null
portah    00E1   Abs   Byte
portb     00E2   Abs   Word
portbh    00E3   Abs   Byte
portbl    00E2   Abs   Byte
portd     0104   Abs   Byte
porti     00D8   Abs   Byte
portp     0152   Abs   Word
portph    0153   Abs   Byte
portpl    0152   Abs   Byte
psw       00C0   Abs   Word
pwmdh     0151   Abs   Byte
pwmdl     0150   Abs   Byte
pwmode    0150   Abs   Word
r1        0184   Abs   Word
r2        0186   Abs   Word
r3        018A   Abs   Word
r4        0142   Abs   Word
r5        0146   Abs   Word
r6        014A   Abs   Word
r7        014E   Abs   Word
rbfl      0001   Abs   Null
rbit9     0003   Abs   Null
rbuf      0124   Abs   Byte
rdrdy     0001   Abs   Null
rdwait    0405   Rel   Null   ROM16
rtccnt    001B   Rel   Byte   BASE
rtcenb    0000   Abs   Null
rtcivl    001A   Rel   Byte   BASE
rtevs     001C   Rel   Byte   BASE
runsys    019F   Rel   Null   ROM16
sio       00D6   Abs   Byte
sk        0006   Abs   Null
slcd      0145   Rel   Null   ROM16
sled      0104   Rel   Null   ROM16
sndbtn    01D2   Rel   Null   ROM16
sndiag    01E4   Rel   Null   ROM16
sndlak    01CA   Rel   Null   ROM16
sndrtc    01C2   Rel   Null   ROM16
so        0005   Abs   Null
sram      00C4   Rel   Null   ROM16
sraml1    00C7   Rel   Null   ROM16
sraml2    00CF   Rel   Null   ROM16
srfsh     008D   Rel   Null   ROM16
sskint    00D2   Rel   Null   ROM16
```

TL/DD/9976-49

```
     stackb   0000   Rel   Word   RAM16
     start    008A   Rel   Null   ROM16
     stmrs    0113   Rel   Null   ROM16
     supi     00A7   Rel   Null   ROM16
     swlast   0017   Rel   Byte   BASE
     swlsnt   0018   Rel   Byte   BASE
     t0ack    0003   Abs   Null
     t0con    0192   Abs   Byte
     t0int    03ED   Rel   Null   ROM16
     t0notp   0365   Rel   Null   ROM16
     t0pdg    035F   Rel   Null   ROM16
     t0pnd    0001   Abs   Null
     t0poll   0359   Rel   Null   ROM16
     t0tie    0000   Abs   Null
     t1       0182   Abs   Word
     t1ack    0007   Abs   Null
     t1int    0367   Rel   Null   ROM16
     t1int1   0370   Rel   Null   ROM16
     t1pnd    0005   Abs   Null
     t1poll   034E   Rel   Null   ROM16
     t1rerr   037E   Rel   Null   ROM16
     t1stp    0006   Abs   Null
     t1tie    0004   Abs   Null
     t2       0188   Abs   Word
     t2ack    0003   Abs   Null
     t2pnd    0001   Abs   Null
     t2stp    0002   Abs   Null
     t2tie    0000   Abs   Null
     t3       018C   Abs   Word
     t3ack    0007   Abs   Null
     t3pnd    0005   Abs   Null
     t3stp    0006   Abs   Null
     t3tie    0004   Abs   Null
     t4       0140   Abs   Word
     t4ack    0003   Abs   Null
     t4out    0000   Abs   Null
     t4pnd    0001   Abs   Null
     t4stp    0002   Abs   Null
     t4tfn    0003   Abs   Null
     t4tie    0000   Abs   Null
     t5       0144   Abs   Word
     t5ack    0007   Abs   Null
     t5out    0004   Abs   Null
     t5pnd    0005   Abs   Null
     t5stp    0006   Abs   Null
     t5tfn    0007   Abs   Null
     t5tie    0004   Abs   Null
     t6       0148   Abs   Word
     t6ack    0003   Abs   Null
     t6int    03A5   Rel   Null   ROM16
```

TL/DD/9976-50

45

```
t6nxt2    03E8    Rel   Null   ROM16
t6nxtc    03B5    Rel   Null   ROM16
t6out     0000    Abs   Null
t6pnd     0001    Abs   Null
t6poll    0353    Rel   Null   ROM16
t6stp     0002    Abs   Null
t6tfn     0003    Abs   Null
t6tie     0000    Abs   Null
t7        014C    Abs   Word
t7ack     0007    Abs   Null
t7out     0004    Abs   Null
t7pnd     0005    Abs   Null
t7stp     0006    Abs   Null
t7tfn     0007    Abs   Null
t7tie     0004    Abs   Null
tbmt      0000    Abs   Null
tbuf      0126    Abs   Byte
tminit    00DF    Rel   Null   ROM16
tmmdh     0191    Abs   Byte
tmmdl     0190    Abs   Byte
tmmode    0190    Abs   Word
tmochk    03A3    Rel   Null   ROM16
tmrint    0348    Rel   Null   ROM16
tmrret    03FE    Rel   Null   ROM16
tmrs      0005    Abs   Null
uart      0006    Abs   Null
upic      00E6    Abs   Byte
upicsv    0000    Rel   Word   BASE
upien     0003    Abs   Null
upiwr     020D    Rel   Null   ROM16
upwret    0343    Rel   Null   ROM16
urdrdy    0007    Abs   Null
uwdone    0000    Abs   Null
uwmode    0001    Abs   Null
uwrrdy    0003    Abs   Null
vbutton   0018    Abs   Null
vdiag     001D    Abs   Null
vlcdak    0017    Abs   Null
vrtc      0011    Abs   Null
wakeup    0002    Abs   Null
wrpgo     0423    Rel   Null   ROM16
wrplp     041D    Rel   Null   ROM16
wrpnl     040A    Rel   Null   ROM16
wrrdy     0000    Abs   Null
xbit9     0005    Abs   Null
xh        00CF    Abs   Byte
xl        00CE    Abs   Byte
xoff      0013    Abs   Null
xon       0011    Abs   Null
xrclk     0003    Abs   Null
```

TL/DD/9976-51

```
xreset    02F0    Rel   Null   ROM16
xtclk     0002    Abs   Null
```

**** Errors: 0, Warnings: 0

TL/DD/9976-52

4.3 Two Demo Programs (NS32CG16 Source Code)

The following two programs run on the NS32CG16 CPU, and exercise the functions implemented in the HPC firmware.

One thing to note in this software is that the interrupt service routines are not written as such; they are simple subroutines called by the actual service routines, which are contained within a modified version of the MON16 monitor program. The reasons for modifying MON16 were two-fold:

1. There is no RAM in the application system within the first 64k of the addressing space. The presence of RAM there is necessary for MON16 to support custom interrupt handlers without internal modification.

2. The HPC requires use of the ''RETT 0'' instruction, rather than ''RETI'', to return from maskable as well as non-maskable interrupts.

Given these two constraints, it was considered most useful to modify MON16 to contain a set of interrupt service routines, which would then use a set of addresses in RAM (a table at address ''vex'') to call custom interrupt servers as standard subroutines. An interrupt service routine calls its custom subroutine after saving the dedicated registers and the general registers, R0, R1 and R2 on the stack.

The symbol ''vex'' is defined externally, and must be declared to match the address used by the modified MON16.

Details of the modified MON16 are available from National Semiconductor Corporation, Microprocessor Applications Group or the Microcontroller Applications Group, phone (408) 721-5000. These modifications are also a standard part of the MONCG monitor program for the NS32CG016 microprocessor.

4.3.1 Panel Exerciser Program

This program for the NS32CG16 CPU exercises several functions of a panel consisting of the following:

- A two-line (8 chars. per line) LCD panel, arranged horizontally into a single 16-line display.
- A speaker, activated by the BEEP command.
- Six pushbuttons, which are presented by the !BUTTON-DATA interrupt to the CPU as follows:

Keyboard Status Byte

0	PB6	PB5	PB4	0	PB2	PB1	PB0

- Five LED's, activated in the SEND-LED command by the following bits:

LED Control Byte

—	—	LD5	LD4	LD3	LD2	LD1	—

The intended layout for the front panel is as shown below. (Please pardon the apparently haphazard assignment of the pushbuttons and LED's; this was dictated by the nature of the module we used for developing this application.)

Front Panel Layout

Cursor Addr. →	00	01	02	03	04	05	06	07	40	41	42	43	44	45	46	47
LCD's:	*			*			*			*			*			*
LED's:	LD2			LD3			LD1			LD5			LD4			(Beep)
PB's:	PB5			PB1			PB0			PB4			PB6			PB2

The locations shown with asterisks on the LCD panel above will display an asterisk character while the corresponding pushbutton below it is depressed. (The number above each LCD location indicates its cursor address in hexadecimal.)

Each time a pushbutton (except PB2) is pressed, the corresponding LED indicator above it is toggled. Rather than toggling an LED, PB2 causes a BEEP command to be issued.

The program starts up the panel with the LCD display blank, and LED's LD1 and LD2 on.

```
1
2                                        # Front Panel Exerciser Program.
3
4                                        # "vex" contains absolute address of NMI service routine entry point.
5                                        # "vex"+4 starts list of maskable interrupt routine entry points;
6                                        #          first is interrupt 0x10.
7
8                                        # Note:  This code assumes that it is running in Supervisor Mode.
9                                        #         Before running, make sure to set PSR to 0200 hex.
10                                       #         Also, all unused interrupts automatically branch to label
11                                       #         "badint";  a breakpoint should be set there.
12
13
14                               .globl  start,main
15                               .globl  rtcint
16                               .globl  lcdint
17                               .globl  swint
18                               .globl  badint
19
20                               .set    hpcctrl,0xFFFC00     # HPC Control/Status I/O location.
21                               .set    hpcdata,0xFFFE00     # HPC Data I/O location.
22                               .set    hpcpoll,0xFD0000     # HPC Poll address (UPIC).
23
24                               .set    INIT,0x0
25                               .set    SET_CONT,0x1
26                               .set    SEND_LCD,0x2
27                               .set    SEND_LED,0x3
28                               .set    BEEP,0x4
29                               .set    RESET_HPC,0xA5
30
31                 start:
32                                                    # Fill interrupt vector locations.
33
34   T00000000  67ddc000        addr    badint,vex           # Interrupt NMI.   (Unimplemented)
                025a0000
                0000
35   T0000000a  67ddc000        addr    badint,vex+4         # Interrupt 0x10.  (Unimplemented)
                02500000
                0004
36   T00000014  67ddc000        addr    rtcint,vex+8         # Interrupt 0x11.  Real-Time Clock.
                021c0000
                0008
37   T0000001e  67ddc000        addr    badint,vex+12        # Interrupt 0x12.  (Unimplemented)
                023c0000
                000c
38   T00000028  67ddc000        addr    badint,vex+16        # Interrupt 0x13.  (Unimplemented)
                02320000
                0010
39   T00000032  67ddc000        addr    badint,vex+20        # Interrupt 0x14.  (Unimplemented)
                02280000
                0014
40   T0000003c  67ddc000        addr    badint,vex+24        # Interrupt 0x15.  (Unimplemented)
                021e0000
                0018
41   T00000046  67ddc000        addr    badint,vex+28        # Interrupt 0x16.  (Unimplemented)
```

TL/DD/9976–53

48

```
                         02140000
                         001c
      42  T00000050      67ddc000              addr     lcdint,vex+32              # Interrupt 0x17.  LCD data written.
                         01e80000
                         0020
      43  T0000005a      67ddc000              addr     swint,vex+36               # Interrupt 0x18.  Pushbutton event.
                         01ea0000
                         0024
      44  T00000064      67ddc000              addr     badint,vex+40              # Interrupt 0x19.  (Unimplemented)
                         01f60000
                         0028
      45  T0000006e      67ddc000              addr     badint,vex+44              # Interrupt 0x1A.  (Unimplemented)
                         01ec0000
                         002c
      46  T00000078      67ddc000              addr     badint,vex+48              # Interrupt 0x1B.  (Unimplemented)
                         01e20000
                         0030
      47  T00000082      67ddc000              addr     badint,vex+52              # Interrupt 0x1C.  (Unimplemented)
                         01d80000
                         0034
      48  T0000008c      67ddc000              addr     badint,vex+56              # Interrupt 0x1D.  Diagnostic:  stop.
                         01ce0000
                         0038
      49  T00000096      67ddc000              addr     badint,vex+60              # Interrupt 0x1E.  (Unimplemented)
                         01c40000
                         003c
      50  T000000a0      67ddc000              addr     badint,vex+64              # Interrupt 0x1F.  (Unimplemented)
                         01ba0000
                         0040
      51  T000000aa      67ddc000              addr     badint,vex+68              # Interrupt 0x20.  (Unimplemented)
                         01b00000
                         0044
      52  T000000b4      67ddc000              addr     badint,vex+72              # Interrupt 0x21.  (Unimplemented)
                         01a60000
                         0048
      53
      54  T000000be      54a500c0              movb     $INIT,hpcctrl    # INITIALIZE command.
                         fffc00
      55  T000000c5      54a500c0              movb     $0,hpcdata       # RTC value:  feature disabled.
                         fffe00
      56
      57  T000000cc      54a503c0              movb     $SEND_LED,hpcctrl      # Initialize LEDs to normal state.
                         fffc00
      58  T000000d3      54a506c0              movb     $0x06,hpcdata
                         fffe00
      59  T000000da      d4a606c0              movb     $0x06,leds       # Save in memory image.
                         000145
      60
      61                              run:
      62  T000000e1      7da30800              bispsrw $0x800           # Enable interrupts from HPC.
      63
      64
      65                              main:                   # Main program starts here.
      66
      67  T000000e5      5cd8c000              movqb    $0,lcdflg        # Set waiting for LCD.
```

                                                                                              TL/DD/9976-54
```

49

```
                     0139
68
69   T000000eb  54a502c0          movb    $SEND_LCD,hpcctrl       # Turn off LCD cursor and clear panel.
                fffc00
70   T000000f2  54a500c0          movb    $0,hpcdata
                fffe00
71   T000000f9  54a502c0          movb    $2,hpcdata
                fffe00
72   T00000100  54a50cc0          movb    $0x0C,hpcdata
                fffe00
73   T00000107  54a501c0          movb    $1,hpcdata
                fffe00
74
75   T0000010e  f4a600c0   l1:    tbitb   $0,lcdflg        # Wait for panel available.
                000110
76   T00000115  9a79              bfc     l1
77
78   T00000117  5cd8c000          movqb   $0,kbdflg
                0104
79
80
81                        kbdlp:
82
83   T0000011d  f4a600c0   l2:    tbitb   $0,kbdflg        # Wait for keyboard data.
                0000fe
84   T00000124  9a79              bfc     l2
85
86   T00000126  7da10800          bicpsrw $0x800           # Sample, and update semaphores.
87   T0000012a  14d8c000          movb    kbdnew,r0
                00f2
88   T00000130  54d8c000          movb    kbdold,r1
                00ed
89   T00000136  d4dec000          movb    kbdnew,kbdold
                00e6c000
                00e7
90   T00000140  5cd8c000          movqb   $0,kbdflg
                00db
91   T00000146  7da30800          bispsrw $0x800
92
93   T0000014a  5f10              movqd   $0,r2            # Initialize offset pointer in r2.
94
95   T0000014c  7800              xorb    r0,r1            # Generate map of differing bits.
96   T0000014e  1c08              cmpqb   $0,r1            # Check that a change actually occurred.
97   T00000150  1a10              bne     lcdlp
98
99   T00000152  54a503c0          movb    $SEND_LED,hpcctrl # If not, error is shown by turning on
                fffc00
100  T00000159  54a520c0          movb    $0x20,hpcdata    #   ALARM LED.
                fffe00
101
102
103                       lcdlp:
104  T00000160  6e8408            ffsb    r1,r2            # Find first differing bit.
105  T00000163  8abfba            bfs     kbdlp            # If none, go wait for another keyboard event.
106  T00000166  4e4810            cbitb   r2,r1            # Clear difference flag.
```

TL/DD/9976–55

```
107
108  T00000169  5cd8c000         movqb  $0,lcdflg      # Do LCD command:  first clear Acknowledge flag.
                00b5
109
110  T0000016f  74a500c0    l3:  tbitb  $0,hpcpoll
                fd0000
111  T00000176  8a79             bfs    l3
112  T00000178  54a502c0         movb   $SEND_LCD,hpcctrl # Start command to display new bit state.
                fffc00
113
114  T0000017f  74a500c0    l4:  tbitb  $0,hpcpoll
                fd0000
115  T00000186  8a79             bfs    l4
116  T00000188  54a502c0         movb   $2,hpcdata     # Flags:  One command followed by one data.
                fffe00
117
118  T0000018f  74a500c0    l5:  tbitb  $0,hpcpoll
                fd0000
119  T00000196  8a79             bfs    l5
120  T00000198  54a502c0         movb   $2,hpcdata     # Two data bytes follow.
                fffe00
121
122  T0000019f  74a500c0    l6:  tbitb  $0,hpcpoll
                fd0000
123  T000001a6  8a79             bfs    l6
124  T000001a8  54e5dac0         movb   lcdloc[r2:b],hpcdata    # Send cursor position byte.
                000078c0
                fffe00
125
126  T000001b3  74a500c0    l7:  tbitb  $0,hpcpoll
                fd0000
127  T000001ba  8a79             bfs    l7
128  T000001bc  3410             tbitb  r2,r0
129  T000001be  8a0c             bfs    l8
130  T000001c0  54a520c0         movb   $0x20,hpcdata  # If new bit is zero, send blank.
                fffe00
131  T000001c7  ea8046           br     lout
132
133  T000001ca  54a52ac0    l8:  movb   $0x2A,hpcdata  # If bit is one, send asterisk instead,
                fffe00
134
135  T000001d1  74a500c0    l9:  tbitb  $0,hpcpoll
                fd0000
136  T000001d8  8a79             bfs    l9
137  T000001da  34a002           tbitb  $2,r0          # and if the key is MENU,
138  T000001dd  9a0b             bfc    l10
139
140  T000001df  54a504c0         movb   $BEEP,hpcctrl  # then beep,
                fffc00
141  T000001e6  ea27             br     lout
142
143  T000001e8  54a503c0   l10:  movb   $SEND_LED,hpcctrl      # else toggle appropriate LED.
                fffc00
144  T000001ef  f8e6dac0         xorb   ledloc[r2:b],leds
                000039c0
```

TL/DD/9976-56

51

```
                          000030
    145  T000001fa  74a500c0    l11:    tbitb   $0,hpcpoll
                          fd0000
    146  T00000201  8a79                bfs     l11
    147  T00000203  54ddc000            movb    leds,hpcdata
                          001cc0ff
                          fe00
    148
    149  T0000020d  f4a600c0    lout:   tbitb   $0,lcdflg       # Wait for LCD Acknowledge interrupt.
                          000011
    150  T00000214  9a79                bfc     lout
    151
    152  T00000216  eabf4a              br      lcdlp           # Go check for any more differing bits.
    153
    154
    155  T00000219  1200                ret     0       # End of main program.
    156
    157
    158                      maindat:        # Data for Main Program.
    159
    160  T0000021b  00      kbdflg: .byte   0       # Keyboard data ready.
    161  T0000021c  00      kbdnew: .byte   0       # New keyboard data (from interrupt service).
    162  T0000021d  00      kbdold: .byte   0       # Saved (previous) keyboard states.
    163  T0000021e  00      lcdflg: .byte   0       # LCD display ready.
    164  T0000021f  00      leds:   .byte   0       # LED states.
    165
    166  T00000220  8683c781    lcdloc: .byte   0x86,0x83,0xC7,0x81,0xC1,0x80,0xC4,0x81
                          c180c481
    167  T00000228  02080000    ledloc: .byte   0x02,0x08,0x0,0x0,0x20,0x04,0x10,0x0
                          20041000
    168
    169
    170
    171                                  # Start of Interrupt Service Routines.
    172                                  # Invoked by ROM interrupt service.  Registers R0..R2 are already
    173                                  #    saved, but no ENTER instruction has been performed yet.
    174                                  #    Because ROM monitor returns using "RETI", we must bypass it
    175                                  #    and return directly with "RETT 0".
    176
    177                      rtcint:         # Interrupt 0x11.  Real-Time Clock.
    178
    179  T00000230  ea2a                br      badint          # UNEXPECTED  (bypass code below)
    180                                                  # Interrupt return procedure:
    181  T00000232  1fb8                cmpqd   $0,tos          #    Discard return address to monitor.
    182  T00000234  72e0                restore [r0,r1,r2]      #    Restore registers saved by monitor.
    183  T00000236  4200                rett    0               #    Return from interrupt directly.
    184
    185                      lcdint:         # Interrupt 0x17.  LCD data written.
    186  T00000238  dcd8ffff            movqb   $1,lcdflg       # Flag that interrupt has occurred.
                          ffe6
    187                                                  # Interrupt return procedure:
    188  T0000023e  1fb8                cmpqd   $0,tos          #    Discard return address to monitor.
    189  T00000240  72e0                restore [r0,r1,r2]      #    Restore registers saved by monitor.
    190  T00000242  4200                rett    0               #    Return from interrupt directly.
    191
```

```
    192                      swint:          # Interrupt 0x18.  Pushbutton event.
    193  T00000244  dcd8ffff            movqb   $1,kbdflg       # Flag that interrupt has occurred.
                          ffd7
    194  T0000024a  d4aec0ff            movb    hpcdata,kbdnew  # Save new keyboard state.
                          fe00ffff
                          ffd2
    195                                                  # Interrupt return procedure:
    196  T00000254  1fb8                cmpqd   $0,tos          #    Discard return address to monitor.
    197  T00000256  72e0                restore [r0,r1,r2]      #    Restore registers saved by monitor.
    198  T00000258  4200                rett    0               #    Return from interrupt directly.
    199
    200                      badint:         # Trap for unimplemented interrupts.  PLACE BREAKPOINT HERE.
    201
    202                                                  # Interrupt return procedure:
    203  T0000025a  1fb8                cmpqd   $0,tos          #    Discard return address to monitor.
    204  T0000025c  72e0                restore [r0,r1,r2]      #    Restore registers saved by monitor.
    205  T0000025e  4200                rett    0               #    Return from interrupt directly.
    206
```

4.3.2 Real-Time Clock Display Program

This program (rtc.s) enables the Real-Time Clock interrupts from the HPC, and counts them to generate a display of elapsed time on the LCD panel.

```
    GNX  Series32000 COFF ASSEMBLER Version  2.5 6/6/88        Page: 1


        1
        2                                   # Real-Time Clock Exerciser:  Places elapsed time in seconds onto
        3                                   #                             LCD Panel.
        4
        5                                   # "vex"   contains absolute address of NMI service routine entry point.
        6                                   # "vex"+4 starts list of maskable interrupt routine entry points;
        7                                   #         first is interrupt 0x10.
        8
        9                                   # Note:  This code assumes that it is running in Supervisor Mode.
       10                                   #        Before running, make sure to set PSR to 0200 hex.
       11                                   #        Also, all unused interrupts automatically branch to label
       12                                   #        "badint";  a breakpoint should be set there.
       13
       14
       15                            .globl  start,main
       16                            .globl  rtcint
       17                            .globl  lcdint
       18                            .globl  swint
       19                            .globl  badint
       20
       21                            .set    hpcctrl,0xFFFFC00    # HPC Control/Status I/O location.
       22                            .set    hpcdata,0xFFFE00     # HPC Data I/O location.
       23                            .set    hpcpoll,0xFD0000     # HPC Poll address (UPIC).
       24                            .set    INIT,0x0
       25                            .set    SET_CONT,0x1
       26                            .set    SEND_LCD,0x2
       27                            .set    SEND_LED,0x3
       28                            .set    BEEP,0x4
       29                            .set    RESET_HPC,0xA5
       30
       31            start:
       32                                                        # Fill interrupt vector locations.
       33
       34  T00000000  67ddc000        addr    badint,vex          # Interrupt NMI.   (Unimplemented)
                      024e0000
                      0000
       35  T0000000a  67ddc000        addr    badint,vex+4        # Interrupt 0x10.  (Unimplemented)
                      02440000
                      0004
       36  T00000014  67ddc000        addr    rtcint,vex+8        # Interrupt 0x11.  Real-Time Clock.
                      02040000
                      0008
       37  T0000001e  67ddc000        addr    badint,vex+12       # Interrupt 0x12.  (Unimplemented)
                      02300000
                      000c
       38  T00000028  67ddc000        addr    badint,vex+16       # Interrupt 0x13.  (Unimplemented)
                      02260000
                      0010
       39  T00000032  67ddc000        addr    badint,vex+20       # Interrupt 0x14.  (Unimplemented)
                      021c0000
                      0014
       40  T0000003c  67ddc000        addr    badint,vex+24       # Interrupt 0x15.  (Unimplemented)
                      02120000
                      0018
       41  T00000046  67ddc000        addr    badint,vex+28       # Interrupt 0x16.  (Unimplemented)
```

53

```
                 02080000
                 001c
42  T00000050    67ddc000        addr    lcdint,vex+32          # Interrupt 0x17.  LCD data written.
                 01e40000
                 0020
43  T0000005a    67ddc000        addr    swint,vex+36           # Interrupt 0x18.  Pushbutton event.
                 01e80000
                 0024
44  T00000064    67ddc000        addr    badint,vex+40          # Interrupt 0x19.  (Unimplemented)
                 01ea0000
                 0028
45  T0000006e    67ddc000        addr    badint,vex+44          # Interrupt 0x1A.  (Unimplemented)
                 01e00000
                 002c
46  T00000078    67ddc000        addr    badint,vex+48          # Interrupt 0x1B.  (Unimplemented)
                 01d60000
                 0030
47  T00000082    67ddc000        addr    badint,vex+52          # Interrupt 0x1C.  (Unimplemented)
                 01cc0000
                 0034
48  T0000008c    67ddc000        addr    badint,vex+56          # Interrupt 0x1D.  Diagnostic:  stop.
                 01c20000
                 0038
49  T00000096    67ddc000        addr    badint,vex+60          # Interrupt 0x1E.  (Unimplemented)
                 01b80000
                 003c
50  T000000a0    67ddc000        addr    badint,vex+64          # Interrupt 0x1F.  (Unimplemented)
                 01ae0000
                 0040
51  T000000aa    67ddc000        addr    badint,vex+68          # Interrupt 0x20.  (Unimplemented)
                 01a40000
                 0044
52  T000000b4    67ddc000        addr    badint,vex+72          # Interrupt 0x21.  (Unimplemented)
                 019a0000
                 0048
53
54  T000000be    54a500c0        movb    $INIT,hpcctrl    # INITIALIZE command.
                 fffc00
55  T000000c5    54a505c0        movb    $5,hpcdata       # RTC value:  interval of 50 milliseconds.
                 fffe00
56
57  T000000cc    5cd8c000        movqb   $0,flags         # Clear interrupt flags.
                 013e
58                               .set    rtcflg,0         # Bit 0 means RTC interrupt detected.
59                               .set    lcdflg,1         # Bit 1 means LCD interrupt detected.
60  T000000d2    d4a614c0        movb    $20,rtcctr       # Clear RTC modulus counter (div by 20).
                 000139
61  T000000d9    5fd8c000        movqd   $0,timcnt        # Clear seconds counter.
                 0133
62
63                       run:
64  T000000df    7da30800        bispsrw $0x800           # Enable interrupts from HPC.
65
66                                        # Neither communication port is selected yet.
67
```

TL/DD/9976-60

```
68                                main:                 # Put main program here.
69
70   T000000e3   4ec8a601            cbitb    $lcdflg,flags   # Place cursor at first character of panel.
                 c0000127
71   T00000eb    54a502c0            movb     $SEND_LCD,hpcctrl
                 fffc00
72   T00000f2    54a500c0            movb     $0,hpcdata
                 fffe00
73   T00000f9    54a501c0            movb     $1,hpcdata
                 fffe00
74   T0000100    54a580c0            movb     $0x80,hpcdata
                 fffe00
75   T0000107    f4a601c0    l1:     tbitb    $lcdflg,flags
                 000103
76   T000010e    9a79                bfc      l1
77
78   T0000110    4ec8a601            cbitb    $lcdflg,flags   # Write initial value of zeroes.
                 c00000fa
79   T0000118    54a502c0            movb     $SEND_LCD,hpcctrl
                 fffc00
80   T000011f    54a5ffc0            movb     $0xFF,hpcdata
                 fffe00
81   T0000126    54a508c0            movb     $8,hpcdata
                 fffe00
82   T000012d    54a530c0            movb     $0x30,hpcdata
                 fffe00
83   T0000134    54a530c0            movb     $0x30,hpcdata
                 fffe00
84   T000013b    54a530c0            movb     $0x30,hpcdata
                 fffe00
85   T0000142    54a530c0            movb     $0x30,hpcdata
                 fffe00
86   T0000149    54a530c0            movb     $0x30,hpcdata
                 fffe00
87   T0000150    54a530c0            movb     $0x30,hpcdata
                 fffe00
88   T0000157    54a530c0            movb     $0x30,hpcdata
                 fffe00
89   T000015e    54a530c0            movb     $0x30,hpcdata
                 fffe00
90   T0000165    f4a601c0    l2:     tbitb    $lcdflg,flags
                 0000a5
91   T000016c    9a79                bfc      l2
92
93   T000016e    f4a600c0    mainlp: tbitb    $rtcflg,flags
                 00009c
94   T0000175    9a79                bfc      mainlp
95   T0000177    4ec8a600            cbitb    $rtcflg,flags
                 c0000093
96
97   T000017f    7ca101              bicpsrb  $0x01           # Clear carry.
98   T0000182    4effa600            addpd    $0x01,timcnt    # Increment BCD elapsed time.
                 000001c0
                 00008a
99
```

55

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.