BUILT-IN TESTS

FOR A

REAL-TIME EMBEDDED SYSTEM

by

PETER ANDREW OLANDER

VOLUME II

Submitted in partial fulfilment of the requirements for the degree of
Master of Science,
in the
Department of Computer Science,
University of Natal
1991
APPENDIX C: SUBSYSTEM-SPECIFIC CODE LISTINGS

1 INTRODUCTION ........................................................................................................ 4

2 SOURCE CODE ........................................................................................................ 5
   2.1 MEASURED RESOURCE UTILISATION ................................................................. 5
   2.2 EXPLANATION OF SOURCE CODE ..................................................................... 5
   2.3 SOURCE CODE LISTINGS .................................................................................... 8
      2.3.1 Real mode code ............................................................................................ 8
         2.3.1.1 PLM86 files .......................................................................................... 8
            2.3.1.1.1 RM_EXEC.PLM ............................................................................. 8
            2.3.1.1.2 RM_APP.PLM .............................................................................. 10
            2.3.1.1.3 RM_BIT.PLM ................................................................................. 23
            2.3.1.1.4 RM_INTS.PLM .............................................................................. 32
            2.3.1.1.5 RM_MSCC.PLM ........................................................................... 37
            2.3.1.1.6 RM_API.PLML ................................................................................ 56
         2.3.1.2 ASM86 files ........................................................................................... 103
            2.3.1.2.1 RM_SEGS.ASM ............................................................................ 103
            2.3.1.2.2 RM_COPY.ASM ........................................................................... 107
         2.3.1.3 Entities publicly declared files ................................................................. 111
            RM_APP.EPD ......................................................................................... 111
            2.3.1.3.2 RM_BIT.EPD ............................................................................... 111
            2.3.1.3.3 RM_INTS.EPD ............................................................................ 111
            2.3.1.3.4 RM_MSCC.EPD ........................................................................... 111
            2.3.1.3.5 RM_API.EPD ................................................................................ 111
            2.3.1.3.6 RM_SEGS.EPD ............................................................................ 111
            2.3.1.3.7 RM_COPY.EPD ............................................................................ 111
         2.3.1.4 Included files ........................................................................................... 117
            2.3.1.4.1 STAT.INC .................................................................................... 117
            2.3.1.4.2 CONSLSerts.INC ......................................................................... 117
            2.3.1.4.3 MSCG.INC ................................................................................... 117
            2.3.1.4.4 CONSMPSC.INC .......................................................................... 117
            2.3.1.4.5 DPRMEM.INC .............................................................................. 117
      2.3.2 Protected mode code ...................................................................................... 125
         2.3.2.1 PL/M 286 files ....................................................................................... 125
            2.3.2.1.1 PM_ENTP.PLML ............................................................................ 125
2.3.2.1.2 PM_EXEC.PLML .......................... 128
2.3.2.1.3 PM_INIT.PLML .......................... 131
2.3.2.1.4 PM_INTS.PLML ........................... 156
2.3.2.1.5 PM_APP.PLML ............................ 159
2.3.2.1.6 PM_PROC2.PLML .......................... 195

2.3.2.2 ASM 286 files ............................... 207
    2.3.2.2.1 PM_SEGS.ASM .......................... 207

2.3.2.3 Entities publicly declared files .................. 212
    2.3.2.3.1 PM_APP.EPD ........................... 212
    2.3.2.3.2 PM_INIT.EPD ........................... 212
    2.3.2.3.3 PM_SEGS.EPD ........................... 213
LIST OF TABLES

Table I : Subsystem-specific code resource utilisation .......................... 5
1 INTRODUCTION

Information pertinent to the subsystem-specific code resident on the system EPROM/RAM cards is presented in the following manner in this appendix:

(a) the measured utilisation of RAM, EPROM and expected maximum stack requirements
(b) a brief explanation of the source code files, together with the full source code listings.

The real mode PL/M and assembler software source files were compiled using the Intel PL/M-86 compiler and Intel 8086/87/88/186 macro assembler, respectively. The Intel 80286 object module files were generated by the PL/M 286 compiler and ASM286 assembler. The same compiler controls used in the compilation of the standardised code were in effect during the generation of both 8086 and 80286 object module file generations, viz. "debug", "large" and "rom".
2 SOURCE CODE

2.1 MEASURED RESOURCE UTILISATION

Table I shows the measured memory utilisation and the maximum stack allocation during the execution of the subsystem-specific code.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Utilisation/allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM</td>
<td>25 kB</td>
</tr>
<tr>
<td>EPROM</td>
<td>88 kB</td>
</tr>
<tr>
<td>Stack</td>
<td>16 kB</td>
</tr>
</tbody>
</table>

2.2 EXPLANATION OF SOURCE CODE

A brief explanation of the function of each software module pertinent to the subsystem-specific code is provided below. The modules are subdivided into the following classes:

(A) Real mode code

(a) PL/M 86 files
(b) ASM 86 files
(c) Entities publicly declared files
(f) Included files

(RM_ <rest of filename>.PLM)
(RM_ <rest of filename>.ASM)
(RM_ <rest of filename>.EPD)
(<filename>.INC)

(B) Protected mode code

(a) PL/M 286 files
(b) ASM 286 files
(c) EPD files
(d) INC files

(PM_ <rest of filename>.PLM)
(PM_ <rest of filename>.ASM)
(PM_ <rest of filename>.EPD)
(<filename>.INC)
(A) Real mode code

(a) PLM86 files

1) RM_EXEC.PLM       Real mode code driving routine
2) RM_APP.PLM        Real mode applications code
3) RM_BIT.PLM        Real mode BIT routines
4) RM_INTS.PLM       Real mode interrupt handlers
5) RM_MSCC.PLM       Mass storage controller demonstration routines
6) RM_API.PLM        Environmental monitoring demonstration routines

(b) ASM 86 files

1) RM_SEGs.ASM       Definition of real mode segments
2) RM_COPY.ASM       Code to copy descriptor tables from ROM to RAM

(d) Entities publicly declared files

1) RM_APP.EPD        Real mode applications code EPD file
2) RM_BIT.EPD        Real mode BIT routines EPD file
3) RM_INTS.EPD       Real mode interrupt handlers EPD file
4) RM_MSCC.EPD       Mass storage controller demonstration routines EPD file
5) RM_API.EPD        Environmental monitoring demonstration routines EPD file
6) RM_SEGs.EPD       Definition of real mode segments EPD file
7) RM_COPY.EPD       Descriptor tables copier EPD file

(e) Included files

1) STAT.INC          Subsystem-specific status latch literals
2) CONSLITS.INC      Literal declarations for environmental monitoring demonstration
3) MSCC.INC          Literal declarations for the mass storage controller demonstration
4) CONSMSPSC.INC     Literal declarations for console specific demonstration
5) DPRMEM.INC        Literal declarations for absolute processor interface locations
(B) Protected mode

(a) PL/M 286 files

(1) PM_EXEC.PLM   Primary processor protected mode executive routine
(2) PM_INIT.PLM   Protected mode initialisation routine
(3) PM_INTS.PLM   Protected mode interrupt service routines
(4) PM_APP.PLM    Protected mode operational demonstration code
(5) PM_ENTP.PLM   Routine to enter protected mode
(6) PM_PROC2.PLM  Secondary processor protected mode executive routine

(b) ASM 286 files

(1) PM_SEGS.ASM   Defines absolute segments for the protected mode code

(c) EPD files

(1) PM_INIT.EPD   EPD file for protected mode initialisation routines
(2) PM_APP.EPD    EPD file for operational demonstration code
(3) PM_SEGS.EPD   EPD file for segment definitions
2.3 SOURCE CODE LISTINGS

2.3.1 Real mode code

2.3.1.1 PLM86 files

2.3.1.1.1 RM_EXEC.PLM

$ include (PLMPAR.INC)

/*******************************************************************************
 * MODULE NAME : REAL_MODE_EXECUTIVE
 *
*******************************************************************************/

* Source Filename : RM_EXEC.PLM
* Source Compiler : PLMb6
* Operating System : DOS 3.10
* Description : Executive driving routine for the real mode applications code.
* Public procedures: None
* EPD files : RM_APP.EPD
* Include files : PLMPAR.INC
*******************************************************************************/

*******************************************************************************/

* HISTORY  Version 1.0 :
* Designed by : P.A. OLANDER  Date : August 1989
* Description : Original
*******************************************************************************/
REAL_MODE_EXECUTIVE: do;

declare SYSTEM_EXECUTIVE label public;

$ include (RM_APP.EPD)
$ eject

SYSTEM_EXECUTIVE:

    call SYSTEM_APPLICATIONS;

end REAL_MODE_EXECUTIVE;
$ include (PLMPAR.INC)

/****************************************************************************
*
* MODULE NAME : REAL_MODE_APPLICATIONS
*
***************************************************************************/

* Source Filename : RM_APP.PLM
*
* Source Compiler : PLM86
*
* Operating System : DOS 3.10
*
* Description : Code resident on the EPROM/RAM card that executes
the real mode system applications code.
*
* Public procedures: None
*
* EPD files : STDIO.EPD
* STDSTAT.EPD
* STDPIC.EPD
* STDBIT.EPD
* STDNS.EPD
* RM_SEGS.EPD
* RM_COPY.EPD
* RM_BIT.EPD
* RM_INTS.EPD
* RM_API.EPD
* RM_MSCE.EPD
*
* Include files : PLMPAR.INC
* LITS.INC
* STAT.INC
*
******************************************************************************/

******************************************************************************/

* HISTORY  Version 1.0 :
*
* Designed by : P.A. OLANDER  Date : August 1989
* Description : Original
*
*******************************************************************************/
REAL_MODE_APPLICATIONS: do;

$ include (.\STD\STDO.EPD)  
$ eject
$ include (.\STD\LITS.INC)  
$ eject
$ include (.\STD\STDSTAT.EPD)  
$ eject
$ include (.\STD\STDPIC.EPD)  
$ eject
$ include (.\STD\STDBIT.EPD)  
$ eject
$ include (.\STD\STDBIT.EPD)  
$ eject
$ include (STAT.INC)  
$ eject
$ include (RM_SEGS.EPD)  
$ eject
$ include (RM_COPY.EPD)  
$ eject
$ include (RM_BIT.EPD)  
$ eject
$ include (RM_INTS.EPD)  
$ eject
$ include (RM_API.EPD)  
$ eject
$ include (RM_MSCC.EPD)  
$ eject
$ include (RM_MSCC.EPD)
$ eject

declare FOREVER literally 'while 1';
declare SECONDO_PROC_HERE literally 'Sadah';
declare MAX_LOOP_COUNTER literally '10';
declare PROC2_FAULTY literally '0a5h';
declare PROC2_PRESEN T literally '0ah';
declare PROC2 literally '0ah';
declare MAX_WORD_COUNT literally '0ffffh';
declare IO_BASE_ADDRESS literally 'Od8h';
declare IO_INTERRUPT_NO literally '80';
declare IO_BAUD_RATE literally '9600';
declare REAL_MODE_MENU (*) byte data
  (ESC, 'l2J', ESC, 'lO;OH',
   'Real Mode Main Demonstration Menu', CR, LF, LF,
   '1) Perform SCA demonstration', CR, LF,
   '2) Perform Mass Storage Unit demonstration', CR, LF, LF, ECM);
The values below initialise the 8274 MPSC as follows:

- **WR4**: Divide by 16 clock; 8-bit sync; 2 stop bits; odd parity
- **WR1**: Disable wait; Wait on transmit; interrupt on all receive
- **WR2A**: Pin-10 is RTS; vectored interrupt; 8086 mode; receive priority
- **WR2B**: This register is the vector table base address
- **WR3**: Receive 8 bits/char; no auto enables; no hunt mode
- **WR5**: DTR on; transmit 8 bits/char; no break; enable transmitter

---

```plaintext
$ eject

INITIALISE_PORT_DATA: procedure;

call INITIALISE_IO_ADDRESSES (PORT_A, IO_BASE_ADDRESS, IO_INTERRUPT_NO);
call INITIALISE_PORT( PORT_A, IO_BAUD_RATE, EIGHT_DATA_BITS, TWO_STOP_BITS, NO_PARITY, TERMINAL);
call INITIALISE_PORT( PORT_B, IO_BAUD_RATE, EIGHT_DATA_BITS, TWO_STOP_BITS, NO_PARITY, TERMINAL);

end INITIALISE_PORT_DATA;

$ eject
```
SET_INTERRUPT_TABLE: procedure;

/******************************************,
* This procedure initially sets all the vectors to point to an illegal
* interrupt handler and then overwrites the appropriate location for each
* vector needed. *
*******************************************/

declare I byte;
declare INTERRUPT_VECTOR(256) pointer at (0);

/* First fill the vector table with a default to the illegal interrupt
handler. */

do I = 0 to 256;
    INTERRUPT_VECTOR (I) = INTERRUPTSPTR (ILLEGAL_INT);
end;

/* Now overwrite the table with appropriate vectors */

call SETS$INTERRUPT (0, DIVIDE_ERROR); /* Intel reserved */
call SETS$INTERRUPT (1, SINGLE_STEP); /* Intel reserved */
call SETS$INTERRUPT (2, NMI); /* Intel reserved */
call SETS$INTERRUPT (3, BREAKPOINT); /* Intel reserved */
call SETS$INTERRUPT (4, INTO_OVERFLOW); /* Intel reserved */
call SETS$INTERRUPT (5, BOUND_RANGE); /* Intel reserved */
call SETS$INTERRUPT (6, INVALID_OPCODE); /* Intel reserved */
call SETS$INTERRUPT (7, PROC_EXT_NOT_AVAILABLE); /* Intel reserved */
call SETS$INTERRUPT (8, DOUBLE_EXCEPTION); /* Intel reserved */
call SETS$INTERRUPT (9, PROC_EXT_SEGMENT_OVERRUN); /* Intel reserved */
call SETS$INTERRUPT (10, INVALID_TASK); /* Intel reserved */
call SETS$INTERRUPT (11, SEGMENT_NOT_PRESENT); /* Intel reserved */
call SETS$INTERRUPT (12, STACK_OVERRUN); /* Intel reserved */
call SETS$INTERRUPT (13, GENERAL_PROTECTION); /* Intel reserved */
call SETS$INTERRUPT (16, NCP_INTERRUPT); /* Intel reserved */
call SETS$INTERRUPT (80, CHB_TX_EMPTY); /* Used by 8274 to transmit chars */

/* Define the Master PIC interrupt handlers */
call SETS$INTERRUPT (96, TMOUT); /* To test off board accesses */
call SETS$INTERRUPT (97, MASTER_CLOCK); /* For Timer 1 & Master PIC tests */
call SETS$INTERRUPT (98, MASTER_INTERRUPT_2);
call SETS$INTERRUPT (99, MASTER_INTERRUPT_3);
call SETS$INTERRUPT (100, MASTER_INTERRUPT_4);

/* Interrupt 5 on the Master PIC is linked to the 8274 and interrupt 6 is
linked to the Slave PIC. Both these devices supply their own vectors. */
call SETSINTERRUPT (103, MASTER_INTERRUPT_7);

/* Define the Slave PIC interrupt handlers */
call SETSINTERRUPT (128, SLAVE_INTERRUPT_0);
call SETSINTERRUPT (129, SLAVE_INTERRUPT_1);
call SETSINTERRUPT (130, SLAVE_INTERRUPT_2);
call SETSINTERRUPT (131, TMAP_INTERRUPT);
/* This interrupt is hardwired to identify the secondary processor */
call SETSINTERRUPT (132, SLAVE_CLOCK); /* For Timer 2 & Slave PIC tests */
call SETSINTERRUPT (133, SLAVE_INTERRUPT_5);
call SETSINTERRUPT (134, SLAVE_INTERRUPT_6);
call SETSINTERRUPT (135, SLAVE_INTERRUPT_7);

end SET_INTERRUPT_TABLE;

$ eject

DETERMINE_STUCK_KEY : procedure;

declare STUCK_MODULE_MASK literally '0f0h';
declare STUCK_KEY_MASK literally '0fh';
declare PATTERN byte;

PATTERN = KEY_STATUS (1) and STUCK_MODULE_MASK; /* get the high nibble */
TEST_RESULT.STUCK_MODULE = ror (PATTERN, 4);
/* 0 for SKM 1 ; 2 for SKM 2 ; 4 for SKM 3 ; 8 for GKSM */
TEST_RESULT.STUCK_KEY = KEY_STATUS (1) and STUCK_KEY_MASK; /* get the low nibble */

end DETERMINE_STUCK_KEY ;

$ eject
MAP_AIS_INITIALIZATION: procedure;

declare LOOP_COUNTER word;

/* Only one processor should initialise the API card */

call SET$INTERRUPT (128, API);
call UNMASK_PIC (SLAVE_INTERRUPT_NO);
call UNMASK_PIC (API_INTERRUPT_NO);
/* First find out if there are any interrupts waiting from the API
  (e.g. stuck keys detected on power-up). */

LOOP_COUNTER = MAX_WORD_COUNT;

do while (API_INTERRUPT = TRUE) or (LOOP_COUNTER > 0);
  if API_INTERRUPT = TRUE then
    do;
      API_INTERRUPT = FALSE;
      LOOP_COUNTER = MAX_WORD_COUNT;
    end;
  else
    LOOP_COUNTER = LOOP_COUNTER - 1;
  end;

call MASK_PIC (API_INTERRUPT_NO);
API_DATA_START_INDEX = 0; /* Reset the buffer indices */
API_DATA_END_INDEX = 0;

call OUTPUT_TO_LATCH (API_DPR_TEST_NO);
call API_RESET_TEST;
call UNMASK_PIC (API_INTERRUPT_NO);
/* Now clear all pending API interrupts */

LOOP_COUNTER = MAX_WORD_COUNT;

do while (API_INTERRUPT = TRUE) or (LOOP_COUNTER > 0);
    if API_INTERRUPT = TRUE then
        do;
            API_INTERRUPT = FALSE;
            LOOP_COUNTER = MAX_WORD_COUNT;
        end;
    else
        LOOP_COUNTER = LOOP_COUNTER - 1;
    endif;
    call TIME (1);
end;

call MASK_PIC (API_INTERRUPT_NO);
API_DATA_START_INDEX = 0; /* Reset the buffer indices */
API_DATA_END_INDEX = 0;

if STUCK_KEY_DETECTED = TRUE then
    call DETERMINE_STUCK_KEY;
else
    do;
        TEST_RESULT.STUCK_MODULE = PASSED;
        TEST_RESULT.STUCK_KEY = PASSED;
    end;
    call TIME (10000); /* Fiddle factor */
end;

call API_SELF_TEST;
call EMAC_TEST;

end MAP_AIS_INITIALIZATION;

$ eject
AIS_INITIALIZATION: procedure public;

declare PROCESSOR_LATE (*) byte data
  (ESC, '12;OH', BEL, BEL, BEL,
   '**** CPU card halted due to late arrival ! ! !',
   CR, LF, EOM);

declare FIRST_HERE (*) byte data
  (ESC, '12;OH', BEL, BEL, BEL,
   'The first processor is here !!',
   CR, LF, EOM);

declare SECOND_HERE (*) byte data
  (ESC, '12;OH', BEL, BEL, BEL,
   'The second processor is here !',
   CR, LF, EOM);

declare HANDSHAKE_RECEIVED (*) byte data
  (ESC, '12;OH', BEL, BEL, BEL,
   'Handshake received from other processor !!',
   CR, LF, EOM);

declare API_CAUSED_HALT (*) byte data
  (ESC, '12;OH', BEL, BEL, BEL,
   '**** Critical error detected : ', CR, LF, LF,
   'System halted due to API card faulty or absent',
   CR, LF, EOM);

declare PROC_HERE word;
declare LOOP_COUNTER word;

API_INTERRUPT = FALSE;
STUCK_KEY_DETECTED = FALSE;
QKBM_KEY_PRESSED = FALSE;
RBM_MOVED = FALSE;
EMAC_INTERRUPT = FALSE;
SPARE_INTERRUPT = FALSE;

call OUTPUT_TO_LATCH (API_DPR_TEST_NO);
call TEST_API_LOCATIONS;
if TEST_RESULT.API_BOARD = ABSENT
or TEST_RESULT.API_BOARD = FAILED then
do;
  if TEST_RESULT.MPSC = PASSED then
    call WRITE_POLL (VDU, API_CAUSED_HALT);
  halt;
end;

$ eject
The first processor to reach this point will perform the API POST and initialisation. Thereafter, the processors will decide who is going to run the primary processor code and who will execute the secondary code. Although the processor ID has already been set for both processors, this code caters for:

1) The primary processor reaching this point moments before the secondary processor.

Since this is the normal operating condition, only the primary processor will initialise the API card and normal roles of execution shall be assumed.

2) The secondary processor reaching this point moments before the primary processor.

In this event, the secondary processor will initialise the API card. Both cards, however, will still assume their normal roles of execution.

3) The primary processor reaching this point long before the secondary processor.

In this case, the primary processor will initialise the API card and then wait for the secondary processor for a pre-determined time. On expiry of this time, the primary processor will assume that the secondary processor is faulty. The former will continue with its normal code execution, reporting the standby processor as faulty. On reaching this point, the secondary processor will issue an appropriate error message and halt execution.

4) The primary processor reaching this point long after the secondary processor.

Here, the secondary processor will perform the appropriate API POST and initialisation and then wait for a pre-determined time for the primary processor to rendezvous. Eventually, the secondary processor will take over the duties of the primary processor. On its late arrival at the rendezvous point, the primary processor will realise that it is at fault, and will report the error and halt.
RUN_ID = MAP;
PROC_HERE = EXCHANGE_FLAG;
if PROC_HERE <> SECOND_PROC_HERE then
    do;
        call WRITE_POLL (VDU, @FIRST_HERE);
    end;
else
    call WRITE_POLL (VDU, @SECOND_HERE);

SYNCHRONISATION_FLAG = PROC_ID;

/* Set global flag to indicate either the primary or secondary processor */
LOOP_COUNTER = 0;

do while (SYNCHRONISATION_FLAG = PROC_ID) and (LOOP_COUNTER < MAX_LOOP_COUNTER);

    /* Wait for a predetermined time (10 ms) for the other processor */
    /* to reset the semaphore. */
    call TIME (10000); /* 1 s delay */
    LOOP_COUNTER = LOOP_COUNTER + 1;
end;

if SYNCHRONISATION_FLAG <> PROC_ID then
    /* Cases (1) and (2) above, i.e. the other processor has reset the */
    /* semaphore, so reinitialise it. */
    do;
        SYNCHRONISATION_FLAG = PROC_ID;
        TMP_FLAG = PROC2_PRESENT;
        STATUS_FLAG = PROC2_PRESENT;
        call WRITE_POLL (VDU, @HANDSHAKE_RECEIVED);
    if PROC_ID = PROC2 then
        do;
            RUN_ID = PROC2;
            call OUTPUT_TO_LATCH (RESET_LATCH);
            /* The secondary processor is to run no further system POST, */
            /* other than reporting its results to global RAM. */
        end;
    end;
end;
else
do;
/* Cases (3) and (4) above, i.e. the counter has expired */
if PROC_ID = MAP then
doi
if $\text{STATUS\_FLAG} = \text{MAP}$ then
/* The secondary processor has taken over as primary processor. */
do;
call OUTPUT\_TO\_LATCH (PROC\_TOO\_LATE);
call WRITE\_POLL (VDU, @PROCESSOR\_LATE);
halt;
end;
else /* The secondary processor is faulty or absent */
$\text{STATUS\_FLAG} = \text{PROC2\_FAULTY}$;
end;
else
do;
/* PROC_ID = PROC2 */
if $\text{STATUS\_FLAG} = \text{PROC2\_FAULTY}$ then
/* The secondary processor took too long, because the */
/* primary processor has already set the flag. */
do;
call OUTPUT\_TO\_LATCH (PROC\_TOO\_LATE);
call WRITE\_POLL (VDU, @PROCESSOR\_LATE);
halt;
end;
else /* The primary processor has taken too long */
do;
/* Pass the necessary handshakes to the API, to enable the API */
/* to perform its required initialisation and testing in */
/* parallel to the dynamic RAM test. */
call MAP\_AIS\_INITIALISATION;
$\text{STATUS\_FLAG} = \text{MAP}$ ;
SYNCHRONISATION\_FLAG = MAP ;
end;
end; /* PROC_ID */
end; /* expired counter */
if PROC_ID = MAP then
    /* Pass the necessary handshakes to the API, to enable the API */
    /* to perform its required initialisation and testing in */
    /* parallel to the dynamic RAM test. */
    call MAP_AIS_INITIALISATION;

    NO_OF_RUNS = SECOND_RUN; /* Set the warm start flag */
    TEST = FALSE; /* Reset the TEST flag to allow normal time-out handling */

    goto COPY_TABLES;

    /* Copies appropriate descriptors, based on PROC_ID, and then jumps */
    /* to a routine that switches the 80286 processor into Protected Mode. */
    /* The rest of the AIS system initialisation and POST is run in */
    /* Protected Mode. */

end AIS_INITIALISATION;

SYSTEM_APPLICATIONS: procedure public;

declare PROC2_HALTED (*) byte data
    (ESC, '[J', ESC, '[O;OH', BEL, BEL, BEL,
     'PROC2 has been reset back into Real Mode and has halted.', EOM);

declare LETTER byte;

enable;

if NO_OF_RUNS <> SECOND_RUN then /* Cold start */
    do;
        TEST = TRUE;
        call AIS_INITIALISATION ;
    end;
else /* NO_OF_RUNS = SECOND_RUN */
/* This implies that a warm start has occurred, so that the real mode
demonstrations should be invoked. These demonstrations consist of
both the console demonstrations and the mass storage controller
card demonstration. */
do;
call OUTPUT_TO_LATCH (RESET_LATCH);
TEST = FALSE;
if PROC_ID <> MAP then
do;
call WRITE_POLL (VDU, @PROC2_HALTED);
halt;
end;
else /* PROC_ID = MAP */
do;
do FOREVER;
call WRITE_POLL (VDU, @REAL_MODE_MENU);
LETTER = INPUT_CHARACTER (KB);
do while (LETTER <> '1') and (LETTER <> '2');
   LETTER = INPUT_CHARACTER (KB);
end;
if LETTER = '1' then /* invoke the console demonstrations */
do;
disable;
call UNMASK_PI (SLAVE_INTERRUPT_NO);
call UNMASK_PI (API_INTERRUPT_NO);
enable;
call CONSOLE_DRIVER;
disable;
call MASK_PI (SLAVE_INTERRUPT_NO);
call MASK_PI (API_INTERRUPT_NO);
call UNMASK_PI (MPSC_INTERRUPT_NO);
/* The interrupts, PI and MPSC were configured differently
during the console demonstrations, so these need to be
reinitialised to the standard configuration. */
call SET_INTERRUPT_TABLE;
call SET$INTERUPT (128, API);
call INITIALISE_PORT_DATA;
enable;
end;
else /* invoke the mass storage controller card demonstrations */
call MASS_STORAGE_TESTS;
end; /* PROC_ID = MAP */
end; /* FOREVER */
end;
end SYSTEM.APPLICATIONS;

end REAL_MODE.APPLICATIONS;
2.3.1.1.3 RM_BIT.PLX

$ include (PLMPAR.INC)

/*****************************************************************************/
* *
* SOURCE FILENAME : RM_BIT.PLX
*
* SOURCE COMPILER : PLMB6
*
* OPERATING SYSTEM : DOS 3.10
*
* DESCRIPTION : Routine to perform the BIT outside of the standard computing segment, prior to entering protected mode.
* *
* PUBLIC PROCEDURES: None
*
* EPD FILES : STDIO.EPD
* STDSTAT.EPD
* STDPIC.EPD
* STDBIT.EPD
* RM_SEGS.EPD
* RM_INTS.EPD
*
* INCLUDE FILES : PLMPAR.INC
* LITS.INC
* STAT.INC
*
*****************************************************************************/

*******************************************************************************/
* *
* HISTORY Version 1.0 :
*
* Designed by : P.A. OLANDER Date : September 1989
*
* Description : Original
*
*******************************************************************************/
REAL_MODE_BIT: do;

$ include (..\STD\STDIO.EPD)
$ eject
$ include (..\STD\STDSTAT.EPD)
$ eject
$ include (..\STD\STDPIC.EPD)
$ eject
$ include (..\STD\STDBIT.EPD)
$ eject
$ include (LITS.INC)
$ eject
$ include (STAT.INC)
$ eject
$ include (RM_SEGS.EPD)
$ eject
$ include (RM_INTS.EPD)
$ eject

declare CLOSED literally '0';
declare OPEN literally '1';
declare OK literally '0';
declare RESET_API literally '30h';
declare ACKNOWLEDGE_CHECKSUM literally '20h';
declare POWER_UP_RESET_ACKNOWLEDGE literally '21h';
declare API_AND_MIDS_RESET_ACKNOWLEDGE literally '22h';
declare ISSUE_API_SELF_TEST_COMMAND literally '40h';
declare API_SELF_TEST_PASSED literally '37h';
declare API_RESET_TIME_MAX literally '20';
declare MAX_API_BYTES literally '07fch';
declare EMAC_SWITCH_MASK literally '0000001b';
RAM_BLOCK_TEST: procedure (BLOCK_BASE_PTR, NO_OF_WORDS) byte public;

declare NO_OF_WORDS word;
declare BLOCK_BASE_PTR pointer;
declare BLOCK_BASE based BLOCK_BASE_PTR word;
declare WORD_TO_TEST_PTR pointer;
declare WORD_TO_TEST based WORD_TO_TEST_PTR word;
declare INDEX word;
declare BLOCK_TEST_RESULT byte;
declare VALUE_TO_SEND word;
declare LOOP_AROUND byte;

BLOCK_TEST_RESULT = PASSED; /* Be optimistic */

/* First check for card presence */
call OFF_BOARD_ACCESS (READ, BLOCK_BASE_PTR, READ);

if CARD_UNDER_TEST = ABSENT then
  BLOCK_TEST_RESULT = ABSENT;
else /* the card is present in the system */
  do;
    VALUE_TO_SEND = WORD_OF_ZERO_ONES;
  do LOOP_AROUND = 1 to 2; /* First write 5555 hex, read it,
    and then write AAAA hex and read it. */

  INDEX = 2; /* Start at second word in block */
/* Write to the RAM */
  do while (BLOCK_TEST_RESULT = PASSED) and (INDEX <= NO_OF_WORDS);
    WORD_TO_TEST_PTR = BUILD$PTR (SELECTOR$OF (BLOCK_BASE_PTR), INDEX);
    call OFF_BOARD_ACCESS (WRITE_IT, 
                       WORD_TO_TEST_PTR, 
                       VALUE_TO_SEND );
  if CARD_UNDER_TEST = ABSENT then
    BLOCK_TEST_RESULT = FAILED;
  INDEX = INDEX + 2;
end;
/* Perform a dummy write to clear the data lines */

call OFF_BOARD_ACCESS (WRITE_IT, BLOCK_BASE_PTR, DUMMY_VALUE);

if CARD_UNDER_TEST = ABSENT then
   BLOCK_TEST_RESULT = FAILED;

/*/ Now read back what was written */

INDEX = 2;

do while (BLOCK_TEST_RESULT = PASSED) and (INDEX <= NO_OF_WORDS);
   WORD_TO_TEST_PTR = BUILD$PTR (SELECTOR$OF (BLOCK_BASE_PTR), INDEX);
   call OFF_BOARD_ACCESS (READ, WORD_TO_TEST_PTR, READ);
   if (CARD_UNDER_TEST = ABSENT) or (ON_BOARD_VAR <> VALUE_TO_SEND) then
      BLOCK_TEST_RESULT = FAILED;
   end;

   INDEX = INDEX + 2;
end;

VALUE_TO_SEND = WORD_OF_ONE_ZEROS;

end; /* LOOP_AROUND 1 to 2 */
end; /* card is present */

return BLOCK_TEST_RESULT;

end RAM_BLOCK_TEST;
TEST_API_LOCATIONS: procedure public;

declare STORE_0 byte;
declare STORE_1 byte;
declare STORE_2 byte;
declare STORE_4 byte;

call OUTPUT_TO_LATCH (API_DPR_TEST_NO);

TEST_RESULT.API_BOARD = PASSED; /* Be optimistic */

/* First check that the card is there, and then save the current values in the first two global locations under test. */

call OFF_BOARD_ACCESS (READ, @API_RAM_BUFFER (0), READ);
if CARD_UNDER_TEST = ABSENT then
    TEST_RESULT.API_BOARD = ABSENT;
else
    do;
        STORE_0 = API_RAM_BUFFER (0);
        STORE_1 = API_RAM_BUFFER (1);
        call OFF_BOARD_ACCESS (READ, @API_RAM_BUFFER (2), READ);
        if CARD_UNDER_TEST = ABSENT then
            TEST_RESULT.API_BOARD = FAILED;
        else
            do;
                STORE_2 = ON_BOARD_VAR;
                call OFF_BOARD_ACCESS (READ, @API_RAM_BUFFER (4), READ);
                if CARD_UNDER_TEST = ABSENT then
                    TEST_RESULT.API_BOARD = FAILED;
                else
                    do;
                        STORE_4 = ON_BOARD_VAR;
                        /* It is safe now to write to the RAM */
                        TEST_RESULT.API_BOARD = RAM_BLOCK_TEST (@API_RAM_BUFFER, 6);
                        /* Now restore the values */
                        API_RAM_BUFFER (0) = STORE_0;
                        API_RAM_BUFFER (1) = STORE_1;
                        API_RAM_BUFFER (4) = STORE_4;
                    end;
                end;
            end;
        end;
    end;
end TEST_API_LOCATIONS;
WAIT_FOR_API: procedure;

declare API_RESET_TIME byte;

API_INTERRUPT = FALSE;
API_RESET_TIME = 0;

call UNMASK_PIC (API_INTERRUPT_NO);

do while (API_INTERRUPT = FALSE) and (API_RESET_TIME <= API_RESET_TIME_MAX);
   API_RESET_TIME = API_RESET_TIME + 1;
   call TIME (10000);
end;

do while API_INTERRUPT = FALSE;
end;

API_INTERRUPT = FALSE; /* Reset the flag */

call MASK_PIC (API_INTERRUPT_NO);

if API_RESET_TIME > API_RESET_TIME_MAX then /* Reset has taken too long */
   TEST_RESULT.API_BOARD = FAILED;
end WAIT_FOR_API;

API_DATA_READ: procedure word;

declare INCOMING_API_DATA word;

INCOMING_API_DATA = API_DATA (API_DATA_START_INDEX);
API_DATA_START_INDEX = (API_DATA_START_INDEX + 1) mod 10;
return INCOMING_API_DATA;
end API_DATA_READ;
API_RESET_TEST: procedure public;

declare API_CHECKSUM word;
declare I word;

call MASK_PIC (API_INTERRUPT_NO);
API_CHECKSUM = 0;
do I = 0 to 9 ;
   API_DATA (I) = Offh;
end;
API_DATA_START_INDEX = 0;
API_DATA_END_INDEX = 0;

AP_WR_DATA = RESET_API; /* Get the API going */
call OUTPUT_TO_LATCH (API_RESET_NO);

TEST_RESULT.API_BOARD = PASSED; /* Be optimistic */
TEST_RESULT.API_RAM = PASSED;
call WAIT_FOR_API;

if TEST_RESULT.API_BOARD = PASSED then
   do;
      do I = 0 to MAX_API_BYTES ;
         API_CHECKSUM = CRC_RESULT (API_CHECKSUM, API_RAM_BUFFER (I));
      end;

      if API_CHECKSUM = API_DATA_READ then
         do;
            AP_WR_DATA = ACKNOWLEDGE_CHECKSUM;
            call OUTPUT_TO_LATCH (API_CHECKSUM_ACK);
         end;
      else
         do;
            TEST_RESULT.API_RAM = FAILED;
            call OUTPUT_TO_LATCH (API_CHECKSUM_NO);
            halt;
         end;
   end;

API_CHECKSUM = API_DATA_READ ;
AP_WR_DATA = ACKNOWLEDGE_CHECKSUM;
call WAIT_FOR_API ; /* Wait for the API to acknowledge the checksum acknowledgement */
if API_DATA_READ <> POWER_UP_RESET_ACKNOWLEDGE then do;
  TEST_RESULT.API_BOARD = FAILED;
  call OUTPUT_TO_LATCH (API_POWER_UP_NO);
  halt;
end;

call WAIT_FOR_API; /* Give the API enough time to inform the AP
  that it and the MIDS has been reset */

if API_DATA_READ <> API_AND_MIDS_RESET_ACKNOWLEDGE then do;
  TEST_RESULT.API_BOARD = FAILED;
  call OUTPUT_TO_LATCH (MIDS_RESET_NO);
  halt;
end;

end API_RESET_TEST;

API_SELF_TEST: procedure public;

call MASK_PIC (API_INTERRUPT_NO);
AP_WR_DATA = ISSUE_API_SELF_TEST_COMMAND;
call WAIT_FOR_API;
if API_DATA_READ <> API_SELF_TEST_PASSED then do;
  TEST_RESULT.API_BOARD = FAILED;
  call OUTPUT_TO_LATCH (API_SELF_TEST_NO);
  halt;
else
  TEST_RESULT.API_BOARD = PASSED;
end API_SELF_TEST;
EMAC_TEST: procedure public;

declare I byte;
declare EMAC_SWITCH_STATUS byte;
declare API_EMAC_LNK byte;
declare EMAC_PRESENT byte;

call OUTPUT_TO_LATCH (EMAC_TEST_NO);
call MASK_PLC (API_INTERRUPT_NO);
EMAC_SWITCH_STATUS = EMAC_STATUS (17);
API_EMAC_LNK = EMAC_STATUS (21);
EMAC_PRESENT = EMAC_STATUS (22);
if (EMAC_PRESENT = OK) and (API_EMAC_LNK = OK) then 
do;
  TEST_RESULT.API_EMAC = PASSED ;
  do I = 0 to 7;
    TEST_RESULT.EMAC_SWITCH (I) = EMAC_SWITCH_STATUS 
        and (rol (EMAC_SWITCH_MASK, I));
  end;
  if TEST_RESULT.EMAC_SWITCH (0) <> CLOSED then 
    TEST_RESULT.EMAC_SWITCH (0) = FAILED;
  else 
    TEST_RESULT.EMAC_SWITCH (0) = PASSED;
  do I = 1 to 7;
    if TEST_RESULT.EMAC_SWITCH (I) <> CLOSED then 
      TEST_RESULT.EMAC_SWITCH (I) = PASSED;
    else 
      TEST_RESULT.EMAC_SWITCH (I) = FAILED;
  end;
  end;
else
  TEST_RESULT.API_EMAC = FAILED ;
end EMAC_TEST;

end REAL_MODE_BIT;
2.3.1.1.4 RM_INTS.PLML

$ include (PLMPAR.INC)

******************************************************************************
* *
* MODULE NAME : REAL_MODE_INTERRUPTS
* *
******************************************************************************
* *
* Source Filename : RM_INTS.PLML
* *
* Source Compiler : PLM86
* *
* Operating System : DOS 3.10
* *
* Description : A suite of interrupt handlers for the real mode demonstration code.
* *
* Public procedures: None
* *
* EPD files : STDIO.EPD
* STDPIC.EPD
* RM_SEGS.EPD
* *
* Include files : PLMPAR.INC
* LITS.INC
* *
******************************************************************************
* *
* HISTORY Version 1.0 :
* *
* Designed by : P.A. OLANDER Date : March 1990
* Description : Original
* *
******************************************************************************

$ eject
C-33

REAL_MODE_INTERRUPTS: DO;

$ include (.\STD\STDIO.EPD)
$ eject
$ include (.\STD\STDPIC.EPD)
$ eject
$ include (LITS.INC)
$ eject
$ include (RM_SEGS.EPD)
$ eject

declare RESERVED_FOR_AP literally '10h';
declare READ_ACKNOWLEDGE literally '90h';
declare CPU_ADDR literally '11000h';
declare CNTRL_REG_8274A literally '0dch'; /* Serial controller write reg */
declare DATA_PORT_8274B literally '0dah'; /* Serial controller data port */

declare KEYBOARD_INTERRUPT byte public;
declare KEYBOARD_INPUT byte public;
declare API_INTERRUPT byte public;
declare DPR_INTERRUPT byte public;
declare STUCK_KEY_DETECTED byte public;
declare QKBM_KEY_PRESSED byte public;
declare RBM_MOVED byte public;
declare EMAC_INTERRUPT byte public;
declare SPARE_INTERRUPT byte public;
declare API_DATA (10) word public;
declare API_DATA_START_INDEX byte public;
declare API_DATA_END_INDEX byte public;
declare DPR_RX byte public;
declare DPR_RX_HI byte public;

declare AP_INTERRUPT_LOCATION word at (CPU_ADDR + 7fch);
CHB_RX: procedure interrupt 82 public;

This interrupt handler is used for serial communications during the API on-line demonstration, rather than the standardised handler. It simply reads a character received from the test terminal and sets an indicator that a character has been received.

KEYBOARD_INPUT = input (DATA_PORT_8274B);  /* read received character */
KEYBOARD_INTERRUPT = TRUE;                   /* set interrupt flag */
output (CNTRL_REG_8274A) = 00111000b;          /* end of interrupt */
call ISSUE_EOI (MPSC_INTERRUPT_NO);           /* non specific EOI - int6 */

end CHB_RX;
API: procedure interrupt 128 public;

declare AP_INTERRUPT_LOCATION byte;

disable;
API_INTERRUPT = TRUE;
API_DATA (API_DATA_END_INDEX) = API_WR_DATA ;
AP_INTERRUPT_LOCATION = API_DATA (API_DATA_END_INDEX) - RESERVED_FOR_AP ;
if (AP_INTERRUPT_LOCATION >= 0) and (AP_INTERRUPT_LOCATION <= 4) then
do;
do case AP_INTERRUPT_LOCATION; /* Now find out what the API wanted to tell the host */
STUCK_KEY_DETECTED = TRUE;
QKBM_KEY_PRESSED = TRUE;
RBM_MOVED = TRUE;
EMAC_INTERRUPT = TRUE;
SPARE_INTERRUPT = TRUE;
end; /* case */
end;
else
if API_DATA (API_DATA_END_INDEX) > 40h then
/* send a read acknowledge to the API */
API_WR_DATA = READ_ACKNOWLEDGE ;

API_DATA_END_INDEX = ( API_DATA_END_INDEX + 1 ) mod 10 ;
call ISSUE_EOI (API_INTERRUPT_NO);
enable;
end API;
DPR_INTERRUPT_READ : procedure interrupt 128 public;

/*************************************************************/
/* Handles interrupts from the API card during the on-line API functionality */
/* demonstration. It reads the location written to by the API and sets a */
/* flag. */
/*************************************************************/

declare
    ADDR_LOC pointer,
    ADDR_RD based ADDR_LOC word;

ADDR_LOC = @AP_INTERRUPT_LOCATION;
DPR_RX = low (ADDR_RD); /* read low byte */
DPR_RX_HI = high (ADDR_RD); /* read high byte */
DPR_INTERRUPT = TRUE; /* set flag */
call ISSUE_EOI (API_INTERRUPT_NO);

end DPR_INTERRUPT_READ;

end REAL_MODE_INTERRUPTS;
RM MSCC.PLM

$ include (PLMPAR.INC)

/*********************************************************************************

* MODULE NAME : MSCC_APPLICATION

*********************************************************************************/

* Source Filename : RM_MSCC.PLM

* Source Compiler : PLM86

* Operating System : DOS 3.10

* Description : Master controlling routines to demonstrate the
   functionality of the mass storage controller card
   and some of its error detection capabilities.

*********************************************************************************/

* Public procedures: MASS_STORAGE_TESTS

* EPD files : STDIO.EPD

* Include files : PLMPAR.INC

*********************************************************************************/

* HISTORY Version 1.0 :

  Designed by : A.J. POLMANS  Date : July 1990

  P.A. OLANDER

  Description : Original

*********************************************************************************/
MSCC_APPLICATION : do;

$ include (MSCC.INC)
$ eject
$ include (STDOUT.EPD)
$ eject

/* Local Constants */

declare TRUE literally '1';
declare FALSE literally '0';
declare FOREVER literally '1';
declare REMOVE_PARITY literally '07fh';

/* Local Variables */

declare VDU byte;
declare KB byte;
declare TEST_OPTION byte;
declare MSCC_PARAMETER_BLOCK (40) byte at (30200h);
declare DATA_BUFFER (5000) byte at (30300h);
declare PROMPT (*) byte data
('Select a letter...
ESC to quit', EOM);
declare ERASE_SCREEN (*) byte data
(ESC, '[2J', ESC, '[0;OH', EOM);

WAIT_FOR_MSCC_TO_COMPLETE : procedure;

/*******************************
* Polls the status register of the MSCC to determine when an interrupt *
* occurs.                        *
*******************************

do while (input (MSCC_STATUS_REGISTER) and 02h) = 00h;
end;

/* Reset Slave Interrupt */

output (MSCC_RESET_SLAVE_INT) = 1;

end WAIT_FOR_MSCC_TO_COMPLETE;
procedure (IN_POINTER, D_WORD_PTR);

/****************************************************
 *                                                 *
 * Converts an incoming word to a double word.      *
 *                                                 *
 ****************************************************/

decall IN_POINTER pointer;
decall IN_WORD (2) word at (IN_POINTER);

decall D_WORD_PTR pointer;
decall OUT_D_WORD based D_WORD_PTR dword;

OUT_D_WORD = dword (IN_WORD (0)) + shl (dword (IN_WORD (1)), 4);

end DOUBLE_WORD_OF;
READ_FILENAME: procedure (FILENAME_PTR);

/*********************************************,~***************************
*                                          *
* Reads the name of a file that is entered from the keyboard and echos  *
* this to the terminal.                     *
*                                          *
*********************************************************************/

declare FILENAME_PTR pointer;
declare FILENAME
    based FILENAME_PTR (*) byte;

declare FILENAME_INDEX byte;
declare FILENAME_CHAR byte;
declare CHAR_ECHO (2) byte;

declare PROMPT_FILENAME (*) byte data
    (CR, LF,
    'Enter a filename : ', EOM);

CHAR_ECHO (1) = EOM;

do FILENAME_INDEX = 0 to 10;
    FILENAME (FILENAME_INDEX) = ' ';  
end;
FILENAME_INDEX = 0;

call WRITE_POLL (VDU, GPROMPT_FILENAME);

FILENAME_CHAR = (INPUT_CHARACTER (KB) and REMOVE_PARITY);
CHAR_ECHO (1) = FILENAME_CHAR;
call WRITE_POLL (VDU, GCHAR_ECHO);

/* Read the filename */
do while (FILENAME_CHAR <> ',' ) and
    (FILENAME_CHAR <> CR ) and
    (FILENAME_INDEX < 8);
/* Convert to uppercase */
if (FILENAME_CHAR >= 'a' ) and
    (FILENAME_CHAR <= 'z' ) then
    FILENAME_CHAR = FILENAME_CHAR - 20h;
/* Add to filename */
FILENAME (FILENAME_INDEX) = FILENAME_CHAR;
FILENAME_INDEX = FILENAME_INDEX + 1;
/* Read the character */
FILENAME_CHAR = (INPUT_CHARACTER (KB) and REMOVE_PARITY);  
CHAR_ECHO (0) = FILENAME_CHAR;  
call WRITE_POLL (VDU, @CHAR_ECHO);  
end;  
/* Read the extension if there is one */  
if FILENAME_CHAR = '.' then  
do;  
FILENAME_INDEX = 8;  
/* Read the character */  
FILENAME_CHAR = (INPUT_CHARACTER (KB) and REMOVE_PARITY);  
CHAR_ECHO (0) = FILENAME_CHAR;  
call WRITE_POLL (VDU, @CHAR_ECHO);  

do while (FILENAME_CHAR <> CR) and  
(FILENAME_INDEX < 11);  
/* Convert to uppercase */  
if (FILENAME_CHAR >= 'a') and  
(FILENAME_CHAR <= 'z') then  
FILENAME_CHAR = FILENAME_CHAR - 20h;  
/* Add to filename */  
FILENAME (FILENAME_INDEX) = FILENAME_CHAR;  
FILENAME_INDEX = FILENAME_INDEX + 1;  
/* Read the character */  
FILENAME_CHAR = (INPUT_CHARACTER (KB) and REMOVE_PARITY);  
CHAR_ECHO (0) = FILENAME_CHAR;  
call WRITE_POLL (VDU, @CHAR_ECHO);  
end;  
end;  
end READ_FILENAME;
FORMAT_FLOPPY_TEST : procedure;

/*******************************************************************************
 *                                                                
 * Issues a command to the MSCC to format a floppy disk and reads the  *
 * status resulting from the format.                                       *
 *******************************************************************************/

declare FORMAT_PAR_BLOCK structure
  (STATUS byte,
    ERROR_MESSAGE_PTR pointer,
    DISK_ID byte,
    DISK_TYPE byte,
    VOLUME_LABEL (11) byte) at (MSCC_PARAMETER_BLOCK);

;/************************************************************************/

;/* Set up parameter block *]/

FORMAT_PAR_BLOCK.DISK_ID = FLOPPY_DISK_O;
FORMAT_PAR_BLOCK.DI SK_TYPE = FLOPPY_360K;
call MOVB (@ ('TEST FORMAT'),
          @FORMAT_PAR_BLOCK.VOLUME_LABEL,
          11);

;/* Set up the command and parameter block address for the MSCC */
/* The write to the GPR interrupts the MSCC to execute the command */
call DOUBLE_WORD_OF (MSCC_PARAMETER_BLOCK,
                    MSCC_PARAMETER_BLOCK_ADDRESS);
output (MSCC_GPR) = FORMAT_DISK;

;/* Wait for command to execute. Then display the error status */
call WAIT_FOR_MSCC_TO_COMPLETE;
call WRITE_POLL (VDU, @ (CR,LF, LF, EOM));
call WRITE_POLL (VDU, FORMAT_PAR_BLOCK.ERROR_MESSAGE_PTR);

end FORMAT_FLOPPY_TEST;

;/************************************************************************/
READ_DIRECTORY_TEST : procedure;

/******************************************************************************
 * * Issues a command to the MSCC to read a disk directory and reads the * *
 * status resulting from this read. * *
*******************************************************************************/

declare DIR_PAR_BLOCK structure
  (STATUS byte,
   ERROR_MESSAGE_PTR pointer,
   DRIVE_ID byte,
   BUFFER_ID dword,
   MAX_BUFFER_LEN dword,
   ACT_BUFFER_LEN dword) at (@MSCC_PARAMETER_BLOCK);

/*-----------------------------------*/
/* Set up the parameter block */

DIR_PAR_BLOCK.DRIVE_ID = FLOPPY_DISK_0;
call DOUBLE_WORD_OF (@DATA_BUFFER, 
  @DIR_PAR_BLOCK.BUFFER_ID);

DIR_PAR_BLOCK.MAX_BUFFER_LEN = 5000;

DIR_PAR_BLOCK.ACT_BUFFER_LEN = 0;

/* Set up the command and parameter block address for the MSCC */
/* The write to the GPR interrupts the MSCC to execute the command */

call DOUBLE_WORD_OF (@MSCC_PARAMETER_BLOCK, 
  @MSCC_PARAMETER_BLOCK_ADDRESS);

output (MSCC_GPR) = READ_DIRECTORY;
/* Wait for command to execute */
call WAIT_FOR_MSCC_TO_COMPLETE;

/* Display data read */
if DIR_PAR_BLOCK.STATUS = OKAY_STATUS then
    DATA_BUFFER(low(DIR_PAR_BLOCK.ACT_BUFFER_LEN)) = EOM;
else if DIR_PAR_BLOCK.STATUS = FILE_TRUNCATED_STATUS then
    DATA_BUFFER(low(DIR_PAR_BLOCK.ACT_BUFFER_LEN) - 1) = EOM;
else
    DATA_BUFFER(0) = EOM;

call WRITE_POLL(VDU, @DATA_BUFFER);

/* Display the error status */
call WRITE_POLL(VDU, @ (CR,LF, LF, EOM));
call WRITE_POLL(VDU, DIR_PAR_BLOCK.ERROR_MESSAGE_PTR);
end READ_DIRECTORY_TEST;
READ_FILE_TEST : procedure;

/******************************************************************************
 * Issues a command to the MSCC to read a file and reads the resulting
 * status.
 *  
 *******************************************************************************

declare READ_PAR_BLOCK structure
    (STATUS byte, 
     ERROR_MESSAGE_PTR pointer, 
     DRIVE_ID byte, 
     FILENAME (11) byte, 
     BUFFER_ID dword, 
     MAX_BUFFER_LEN dword, 
     ACT_BUFFER_LEN dword) at (GMSCC_PARAMETER_BLOCK);

declare FILENAME (11) byte;

/******************************************************************************
 * Set up parameter block */

READ_PAR_BLOCK.DRIVE_ID = FLOPPY_DISK_0;
call DOUBLE_WORD_OF (@DATA_BUFFER, @READ_PAR_BLOCK.BUFFER_ID);
READ_PAR_BLOCK.MAX_BUFFER_LEN = 5000;
READ_PAR_BLOCK.ACT_BUFFER_LEN = 0;
call READ_FILENAME (@FILENAME);
call MOVB (@FILENAME, @READ_PAR_BLOCK.FILENAME, 11);

/******************************************************************************
 * Set up the command and parameter block address for the MSCC */
 /* The write to the GPR interrupts the MSCC to execute the command */
call DOUBLE_WORD_OF (@GMSCC_PARAMETER_BLOCK, 
                       @GMSCC_PARAMETER_BLOCK_ADDRESS);
output (MSCC_GPR) = READ_FILE;
/* Wait for command to execute */
call WAIT FOR MSCC TO COMPLETE;

/* Display data read */

if READ_PAR_BLOCK.STATUS = OKAY_STATUS then
    DATA_BUFFER (low (READ_PAR_BLOCK.ACT_BUFFER_LEN)) = EOM;
else if READ_PAR_BLOCK.STATUS = FILE_TRUNCATED_STATUS then
    DATA_BUFFER (low (READ_PAR_BLOCK.ACT_BUFFER_LEN) - 1) = EOM;
else
    DATA_BUFFER (0) = EOM;
end if;
call WRITE_POLL (VDU, &DATA_BUFFER);

/* Display the error status */
call WRITE_POLL (VDU, @ (CR, LF, LF, EOM));
call WRITE POLL (VDU, READ_PAR_BLOCK.ERROR MESSAGE_PTR);
end READ_FILE_TEST;
WRITE_FILE_TEST : procedure;

declare WRITE_PAR_BLOCK structure (STATUS byte,
ERROR_MESSAGE_PTR pointer,
DRIVE_ID byte,
FILENAME (11) byte,
BUFFER_ID dword,
MAX_BUFFER_LEN dword,
ACT_BUFFER_LEN dword) at (@MSCC_PARAMETER_BLOCK);

declare DATA_TO_BE_WRITTEN (*) byte
  data ('ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789', CR, LF,
       'abcdefghijklmnopqrstuvwxyz')@#$%A&*(' CR, LF);

/* ---------------------------------------------------------- */
/* Set up the parameter block */
WRITE_PAR_BLOCK.DRIVE_ID = FLOPPY_DISK_0;
call DOUBLE_WORD_OF (@DATA_TO_BE_WRITTEN,
                     @WRITE_PAR_BLOCK.BUFFER_ID);
WRITE_PAR_BLOCK.MAX_BUFFER_LEN = 5000;
WRITE_PAR_BLOCK.ACT_BUFFER_LEN = length (DATA_TO_BE_WRITTEN);
call MOVB (@('ALPHA', BET'),
          @WRITE_PAR_BLOCK.FILENAME,
          11);
/* Set up the command and parameter block address for the MSCC */
/* The write to the GPR interrupts the MSCC to execute the command */
call DOUBLE_WORD_OF (@MSCC_PARAMETER_BLOCK,
                     @MSCC_PARAMETER_BLOCK_ADDRESS);
output (MSCC_GPR) = WRITE_FILE;
/* Wait for command to execute. Then display the error status */
call WAIT_FOR_MSCC_TO_COMPLETE;
call WRITE_POLL (VDU, @ (CR,LF, LF, ECM));
call WRITE_POLL (VDU, WRITE_PAR_BLOCK.ERROR_MESSAGE_PTR);
end WRITE_FILE_TEST;
PRINT_FILE_TEST : procedure;

/*********************************************~r***************************
* * Issues a command to the MSCC to print a file and reads the resulting *
* * status. *
* *
*********************************************~**************************/

declare PRINT_PAR_BLOCK structure
    (STATUS byte,
    ERROR_MESSAGE_PTR pointer,
    DRIVE_ID byte,
    FILENAME (11) byte,
    PRINTER_ID byte) at (CilMSCC_PARAMETER_BLOCK);

declare FILENAME (11) byte;

/*----------------------------------------------------------*/
/* Set up the parameter block */
PRINT_PAR_BLOCK.PRINTER_ID = PRINTER_PORT;
PRINT_PAR_BLOCK.DRIVE_ID = FLOPPY_DISK_0;

call READ_FILENAME (@FILENAME);
call MOVB (@FILENAME, @PRINT_PAR_BLOCK.FILENAME, 11);

/* Set up the command and parameter block address for the MSCC */
/* The write to the GPR interrupts the MSCC to execute the command */
call DOUBLE_WORD_OF (@MSCC_PARAMETER_BLOCK, 
                      @MSCC_PARAMETER_BLOCK_ADDRESS);
output (MSCC_GPR) = PRINT_FILE;

/* Wait for command to execute. Then display the error status */
call WAIT_FOR_MSCC_TO_COMPLETE;
call WRITE_POLL (VDU, @ (CR,LF, LF, EOM));
call WRITE_POLL (VDU, PRINT_PAR_BLOCK.ERROR_MESSAGE_PTR);

end PRINT_FILE_TEST;
COPY_FILE_TEST : procedure;

/*******************************************************************************/
/*
* Issues a command to the MSCC to copy a file from a source to a
* destination and reads the status resulting from this copy.
* 
*******************************************************************************/

declare COPY_PAR_BLOCK structure

(STATUS byte,
ERROR_MESSAGE_PTR pointer,
SOURCE_DRIVE_ID byte,
SOURCE_FILENAME (11) byte,
DEST_DRIVE_ID byte,
DEST_FILENAME (11) byte) at (@MSCC_PARAMETER_BLOCK);

declare FILENAME (11) byte;

/*******************************************************************************/
/* Set up the parameter block */
COPY_PAR_BLOCK.SOURCE_DRIVE_ID = FLOPPY_DISK_O;
COPY_PAR_BLOCK.DEST_DRIVE_ID = FLOPPY_DISK_O;

call READ_FILENAME (@FILENAME);
call MOVB (@FILENAME,
   @COPY_PAR_BLOCK.SOURCE_FILENAME,
   11);

call READ_FILENAME (@FILENAME);
call MOVB (@FILENAME,
   @COPY_PAR_BLOCK.DEST_FILENAME,
   11);

/*******************************************************************************/
/* Set up the command and parameter block address for the MSCC */
/* The write to the GPR interrupts the MSCC to execute the command */
call DOUBLE_WORD_OF (@MSCC_PARAMETER_BLOCK,
   @MSCC_PARAMETER_BLOCK_ADDRESS);
output (MSCC_GPR) = COPY_FILE;

/*******************************************************************************/
/* Wait for command to execute. Then display the error status */
call WAIT_FOR_MSCC_TO_COMPLETE;
call WRITE_POLL (VDU, @ (CR, LF, LF, EOM));
call WRITE_POLL (VDU, COPY_PAR_BLOCK.ERROR_MESSAGE_PTR);

end COPY_FILE_TEST;
DELETE_FILE_TEST : procedure;

/*******************************************************************************
 *                          
 *  Issues a command to the MSCC to delete a file and reads the resulting  
 *  status.                  
 *                          
******************************************************************************* /

declare DELETE_PAR_BLOCK structure
    (STATUS byte,
     ERROR_MESSAGE_PTR pointer,
     DRIVE_ID byte,
     FILENAME (11) byte) at (MSCC_PARAMETER_BLOCK);

declare FILENAME (11) byte;

/*----------------------------------------------------------* /
/* Set up the parameter block */

DELETE_PAR_BLOCK.DRIVE_ID = FLOPPY_DISK_D;

call READ_FILENAME (GFILENAME);
call MOVB (GFILENAME,
        GDELETE_PAR_BLOCK.FILENAME,
        11);

/* Set up the command and parameter block address for the MSCC */
/* The write to the GPR interrupts the MSCC to execute the command */
call DOUBLE_WORD_OF (GMSCC_PARAMETER_BLOCK,
        MSCC_PARAMETER_BLOCK_ADDRESS);
output (MSCC_GPR) = DELETE_FILE;

/* Wait for command to execute. Then display the error status */
call WAIT_FOR_MSCC_TO_COMPLETE;
call WRITE_POLL (VDU, @ (CR, LF, LF, EOM));
call WRITE_POLL (VDU, DELETE_PAR_BLOCK.ERROR_MESSAGE_PTR);

end DELETE_FILE_TEST;
WRITE_LARGE_FILE: procedure;

/*********************************************~'***************************
* * Issues a command to the MSCC to write a file that is larger than 64K *
* * and reads the status response. The test is expected to result in the *
* * file being truncated. *
* *
*********************************************,r**************************/ declare WRITE_PAR_BLOCK structure
  (STATUS byte,
   ERROR_MESSAGE_PTR pointer,
   DRIVE_ID byte,
   FILENAME (11) byte,
   BUFFER_ID dword,
   MAX_BUFFER_LEN dword,
   ACT_BUFFER_LEN dword) at (@MSCC_PARAMETER_BLOCK);

declare DATA_TO_BE_WRITTEN (*) byte
data ('ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789', CR, LF,
      'abcdefghijklmnopqrstuvwxyz', CR, LF);

/*----------------------------------------------------------*/
/* Set up the parameter block */
WRITE_PAR_BLOCK.DRIVE_ID = FLOPPY_DISK_C;
call DOUBLE_WORD_OF (@DATA_TO_BE_WRITTEN,
                     @WRITE_PAR_BLOCK.BUFFER_ID);
WRITE_PAR_BLOCK.MAX_BUFFER_LEN = 5000h;
WRITE_PAR_BLOCK.ACT_BUFFER_LEN = 10000h; /* Exceed the allowed length */
call MOVB @('ALPHA MAX'),
     @WRITE_PAR_BLOCK.FILENAME,
     11);

/*----------------------------------------------------------*/
/* Set up the command and parameter block address for the MSCC */
/* The write to the GPR interrupts the MSCC to execute the command */
call DOUBLE_WORD_OF (@MSCC_PARAMETER_BLOCK,
                     @MSCC_PARAMETER_BLOCK_ADDRESS);
output (MSCC_GPR) = WRITE_FILE;
/* Wait for command to execute. Then display the error status */
call WAIT_FOR_MSCC_TO_COMPLETE;
call WRITE_POLL (VDU, @ (CR,LF, LF, EOM));
call WRITE_POLL (VDU, WRITE_PAR_BLOCK.ERROR_MESSAGE_PTR);
/* Should respond with an error status */
end WRITE_LARGE_FILE;
SIMULATE_FAULT : procedure;

*****************************************************************************
* Simulate a faulty condition during on-line testing of the mass storage controller card.
*****************************************************************************

declare PORT_DATA structure
  (STATUS byte,
   ERROR_MESSAGE_PTR pointer,
   PORT_ID byte,
   BAUD_RATE byte,
   NO_OF_DATA_BITS byte,
   NO_OF_STOP_BITS byte,
   PORT_MODE byte,
   END_CHARACTER byte,
   BLOCK_SIZE byte,
   SERIAL_BUFFER_PTR pointer) at (SMSCC_PARAMETER_BLOCK);

declare FAUlT_SIM_MENU (*) byte data
  ('Mass storage fault simulation menu', CR, LF,
   'A Write a file larger than 64K', CR, LF,
   'B Issue an invalid baud rate', CR, LF,
   'C Issue an invalid number of data bits', CR, LF,
   'D Issue an invalid port number', CR, LF,
   'E Issue an invalid port mode', CR, LF, EOM);

declare TEST_STOP byte;

TEST_STOP = FALSE;
PORT_DATA.PORT_ID = SERIAL_PORT_3;
PORT_DATA.BAUD_RATE = 9600;
PORT_DATA.NO_OF_DATA_BITS = 8;
PORT_DATA.NO_OF_STOP_BITS = 2;
PORT_DATA.PORT_MODE = NO_INTERRUPTS or
                      NO_HANDSHAKING or
                      MONITOR_TERMINAL;
PORT_DATA.END_CHARACTER = EOM;
do while TEST_STOP = FALSE;
    /* Display the menu */

call WRITE_POLL (VDU, ERASE_SCREEN);
call WRITE_POLL (VDU, FAULT_SIM_MENU);
call WRITE_POLL (VDU, PROMPT);

    /* Wait for an option to be selected and unmask the parity bit */

TEST_OPTION = INPUT_CHARACTER (KB);

if TEST_OPTION = ESC then
    TEST_STOP = TRUE;
else
    do;
        TEST_OPTION = (TEST_OPTION) and (REMOVE_PARITY);
    end;

    /* Perform the test if a valid option was selected */

if (TEST_OPTION >= 'A') and (TEST_OPTION <= 'E') then
    do;
        do case TEST_OPTION - 'A';
            call WRITE_LARGE_FILE; /* A */
            PORT_DATA.BAUD_RATE = 0; /* B */
            PORT_DATA.NO_OF_DATA_BITS = 0; /* C */
            PORT_DATA.PORT_ID = Offh; /* D */
            PORT_DATA.PORT_MODE = Offh; /* E */
        end; /* case */
        if TEST_OPTION <> 'A' then
            do;
                call DOUBLE_WORD_OF (MSCC_PARAMETER_BLOCK,
                        MSCC_PARAMETER_BLOCK_ADDRESS);
                output (MSCC_GPR) = INITIALISE_SERIAL_PORT;
            end;
        end;
    end; /* (TEST_OPTION >= 'A') and (TEST_OPTION <= 'E') */
end; /* if TEST_STOP = FALSE */
end; /* while */

end SIMULATE_FAULT;
MASS_STORAGE_TESTS : procedure public;

/****************************************************************************
* Demonstrate some of the functional features of the mass storage
* controller card together with its software-implemented error
* detection capabilities.
*/

declare MENU (*) byte data
('Mass storage demonstration menu', CR, LF,
    'A Format floppy disk', CR, LF,
    'B Read directory', CR, LF,
    'C Read a file', CR, LF,
    'D Copy a file', CR, LF,
    'E Delete a file', CR, LF,
    'F Write a new file', CR, LF,
    'G Fault simulation menu', CR, LF, EOM);

declare TEST_STOP byte;

VDU = PORT_B;
KB = PORT_B;
TEST_STOP = FALSE;

do while TEST_STOP = FALSE;
    /* Display the menu */
    call WRITE_POLL (VDU, ERASE_SCREEN);
    call WRITE_POLL (VDU, AMENU);
    call WRITE_POLL (VDU, GPROMPT);
/* Wait for an option to be selected and unmask the parity bit */

TEST_OPTION = INPUT_CHARACTER (KB);

if TEST_OPTION = ESC then
    TEST_STOP = TRUE;
else
    do;

    TEST_OPTION = (TEST_OPTION) and (REMOVE_Fparity);

end; /* while loop */

end MSCC_APPLICATION;

/* Wait for an option to be selected and unmask the parity bit */

TEST_OPTION = INPUT_CHARACTER (KB);

if TEST_OPTION = ESC then
    TEST_STOP = TRUE;
else
    do;

    TEST_OPTION = (TEST_OPTION) and (REMOVE_Fparity);

end; /* while loop */

end MSCC_APPLICATION;

if (TEST_OPTION >= 'A') and 
   (TEST_OPTION <= 'G') then
    do case (TEST_OPTION - 'A');
        call FORMAT_FLOPPY_TEST; /* A */
        call READ_DIRECTORY_TEST; /* B */
        call READ_FILE_TEST; /* C */
        call COPY_FILE_TEST; /* D */
        call DELETE_FILE_TEST; /* E */
        call WRITE_FILE_TEST; /* F */
        call SIMULATE_FAULT; /* G */
    end;

    /* Pause to display the command error status */

    call TIME (100000);
end; /* TEST_STOP = TRUE */

end; /* while loop */

end MSCC_APPLICATION;

end MASS_STORAGE_TESTS;

end MSCCAPPLICATION;
2.3.1.6 RM_API.PLM

$ include (PLMPAR.INC)

/*********************************************

* MODULE NAME : API_APPLICATIONS
*
*********************************************

* Source Filename : RM_API.PLM
* Source Compiler : PLM86
* Operating System : DOS 3.10
* Description : Executes the applications code of the applications processor interface and environmental monitoring and control demonstrations.
* Public procedures: CONSOLE_DRIVER

* EPD files : STDIO.EPD
* STDPIC.EPD
* STDCNVST.EPD
* RM_INTS.EPD
* RM_BIT.EPD
* RM_SEG.EPD
* Include files : PLMPAR.INC
* CONSITNS.INC
* CONSMPSC.INC
* DPRMEM.INC

******************************************************************************
******************************************************************************

* HISTORY Version 1.0 :

* Designed by : N.H. MEHTA Date : October 1989
* P.A. OLANDER
* Description : Original

******************************************************************************
API_APPLICATIONS:
   do;

   $ include (STIO.EPD)
   $ eject
   $ include (CONSLITS.INC)
   $ eject
   $ include (CONSMPS.INC)
   $ eject
   $ include (DPRMEM.INC)
   $ eject
   $ include (STDPIC.EPD)
   $ eject
   $ include (STDCNV.T.EPD)
   $ eject
   $ include (RM_INTS.EPD)
   $ eject
   $ include (RM_BIT.EPD)
   $ eject
   $ include (RM_SEGS.EPD)
   $ eject

   /* Global declarations */
   declare MENU_PTR pointer;
   declare DPR_CONTROL pointer;
   declare BASE_DPR pointer;
   declare MIDS_RESET_CODE byte;
   declare CHANNEL_PASS_CODE byte;
   declare BASE_CODE byte;
   declare RBM_X_VAL byte;
   declare RBM_Y_VAL byte;

   declare PROMPT (*) byte data
      (CR, LF, LF,
       'Select a letter... ESC to quit', EOM);

   declare WAIT (*) byte data
      (DIS_HME, RES_AREA, ' WAIT (Press ESC to Abort)', EOM);

   declare REP_ESC (*) byte data
      (DIS_HME, POINT_AREA, LF, CLR_SCREEN, EOM);

   declare ERASE_NEXT (*) byte data
      (CLR_SCREEN, EOM);

   declare NEXT_LINE (*) byte data
      (CR, LF, EOM);

   declare TEST_MESS (*) byte data
      (CR, LF, ' Test Selected - ', EOM);
INITIALISE_INTERRUPT_VECTORS: procedure ;

    call set$interrupt (82, CHB_RX);
call set$interrupt (128, DPR_INTERRUPT_READ);

end INITIALISE_INTERRUPT_VECTORS;

INITIALISE_VARIABLES : procedure ;

    KEYBOARD_INTERRUPT = FALSE;
    KEYBOARD_INPUT = FALSE;
    DPR_INTERRUPT = FALSE;
    DPR_RX = 0;
    DPR_RX_HI = 0;
    RBM_X_VAL = 0;
    RBM_Y_VAL = 0;

end INITIALISE_VARIABLES;

CONSOLE_PIC_INITIALISATION: procedure ;

/**************************************************************
**                                                          
**  Initialises the 8259A PIC for the console demonstrations 
**                                                          
**************************************************************/

call INITIALISE_8259A_PIC; /* Standardised initialisation */

/* Master */

output( PIC_MASTER_8259A_ADR1) = NO_POLL;
/* OCW_3 no poll command */

/* Slave */

output( PIC_SLAVE_8259A_ADR1 ) = NO_POLL;
/* OCW_3 no poll command */

call UNMASK_PIC (MPSC_INTERRUPT_NO); /* All interrupts initially masked */
call UNMASK_PIC (SLAVE_INTERRUPT_NO); /* so enable API and keyboard */
call UNMASK_PIC (API_INTERRUPT_NO); /* interrupts */

end CONSOLE_PIC_INITIALISATION;
CONSOLE_MPSC_INITIALISATION: procedure;

/**************************
  Initialises the 8274 MPSC for the console demonstrations:
  both channels are initialised to perform asynchronous operations.

****************************/

WRITE_TO_8274: procedure (CHANNEL, REG, VALUE);

declare (CHANNEL, REG, VALUE) byte;

if REG = 0 then
  output (CHANNEL) = VALUE;
else
  do;
  output (CHANNEL) = REG;
  output (CHANNEL) = VALUE;
  end;

  call TIME(10);

end WRITE_TO_8274;

call WRITE_TO_8274 (CNTRL_REG_8274A, 0, CHANNEL_RESET);
call WRITE_TO_8274 (CNTRL_REG_8274B, 0, CHANNEL_RESET);
call WRITE_TO_8274 (CNTRL_REG_8274A, 4, REG_4_VAL);
call WRITE_TO_8274 (CNTRL_REG_8274B, 4, REG_4_VAL);
call WRITE_TO_8274 (CNTRL_REG_8274A, 1, REG_1A_VAL);
call WRITE_TO_8274 (CNTRL_REG_8274A, 2, REG_2A_VAL);
call WRITE_TO_8274 (CNTRL_REG_8274A, 5, REG_5A_VAL);
call WRITE_TO_8274 (CNTRL_REG_8274A, 3, REG_3A_VAL);
call WRITE_TO_8274 (CNTRL_REG_8274B, 1, REG_1B_VAL);
call WRITE_TO_8274 (CNTRL_REG_8274B, 2, REG_2B_VAL);
call WRITE_TO_8274 (CNTRL_REG_8274B, 5, REG_5B_VAL);
call WRITE_TO_8274 (CNTRL_REG_8274B, 3, REG_3B_VAL);

end CONSOLE_MPSC_INITIALISATION;
WRITE_OUT: procedure (MESSAGE_PTR);

/*******************************************************************************/
* 
* Disables interrupts, sends a message pointed to by MESSAGE_PTR to the 
* terminal and then enables interrupts again. 
* 
/*******************************************************************************/
declare MESSAGE_PTR pointer;
declare MESSAGE based MESSAGE_PTR (1000h) byte;
declare DELAY_WR byte;
declare NEW_PTR word;
declare I word;
call MASK_PIC (MPSC_INTERRUPT_NO);
call MASK_PIC (SLAVE_INTERRUPT_NO);
call MASK_PIC (API_INTERRUPT_NO);
call WRITE_POLL (VDU, MESSAGE_PTR);
call UNMASK_PIC (MPSC_INTERRUPT_NO);
call UNMASK_PIC (SLAVE_INTERRUPT_NO);
call UNMASK_PIC (API_INTERRUPT_NO);
end WRITE_OUT;

SEND_CHARACTER: procedure (CHARACTER);

/*******************************************************************************/
* 
* Disables interrupts, sends a single character to the terminal and then 
* enables interrupts again. 
* 
/*******************************************************************************/
declare CHARACTER byte;
declare STRING (2) byte;
STRING (1) = EOM;
call MASK_PIC (MPSC_INTERRUPT_NO);
call MASK_PIC (SLAVE_INTERRUPT_NO);
call MASK_PIC (API_INTERRUPT_NO);
call WRITE_POLL (VDU, @STRING);
call UNMASK_PIC (MPSC_INTERRUPT_NO);
call UNMASK_PIC (SLAVE_INTERRUPT_NO);
call UNMASK_PIC (API_INTERRUPT_NO);
end SEND_CHARACTER;
ESCAPE : procedure byte ;

/*****************************
 * 
 * Determines whether an <Escape> character was received. 
 * 
**********************************/

declare CONTINUE byte;

do while KEYBOARD_INTERRUPT = FALSE;
    end;
KEYBOARD_INTERRUPT = FALSE;
if KEYBOARD_INPUT = ESC then
    CONTINUE = FALSE;
else
    CONTINUE = TRUE;

return CONTINUE;

end ESCAPE;
READ_DPR : procedure byte;

/*******************
 * Returns the value read in the API interrupt handler.
 *******************/

declare DPR_VALUE byte;
declare ABORT_MESSAGE (*) byte data
  (DIS_HME, RES_AREA, CLR_SCREEN,
   ' Test Aborted ',
   EOM);

do while (DPR_INTERRUPT = FALSE) and (KEYBOARD_INTERRUPT = FALSE);
/* wait for valid interrupt */
end;

if (DPR_INTERRUPT = FALSE) and (KEYBOARD_INTERRUPT = TRUE) then
do;
call WRITE_OUT (ABORT_MESSAGE (0));
DPR_VALUE = ABORTED;
KEYBOARD_INTERRUPT = FALSE;
end;

if (DPR_INTERRUPT = TRUE) then
do;
DPR_VALUE = DPR_RX; /* read */
DPR_INTERRUPT = FALSE;
end;

return DPR_VALUE;

end READ_DPR;
READ_DPR_BUFFER : procedure (BUFFER_PTR) byte;

/******************************************************************************
 *
 * Returns the value read into the dual port RAM buffer.
 *
 ***************************************************/

declare BUFFER_DATA byte;
declare BUFFER_PTR pointer;
declare BUFFER_VALUE based BUFFER_PTR byte;

BUFFER_DATA = BUFFER_VALUE;
return BUFFER_DATA;
end READ_DPR_BUFFER;

WRITE_DPR : procedure (DPR_LOCATION, DPR_VALUE);

/******************************************************************************
 *
 * Writes to a location in dual port RAM.
 *
 ***************************************************/

declare DPR_LOCATION pointer;
declare DPR_DATA based DPR_LOCATION byte;
declare DPR_VALUE byte;

DPR_DATA = DPR_VALUE;
end WRITE_DPR;
WRITE_RESULT : procedure (ACTUAL_VALUE, EXPECTED_VALUE) ;

/***************************************************************************/
/* */
/* Compares an actual result to an expected result and outputs the */
/* appropriate message to the terminal. */
/* */
办法***************************************************************************/

declare ACTUAL_VALUE byte;
declare EXPECTED_VALUE byte;
declare PASS (*) byte data (DIS_HME,POINT_AREA,
  'Test Passed - ', CLR_SCREEN, EOM);
declare FAIL (*) byte data (DIS_HME,POINT_AREA,
  'Test Failed - ', CLR_SCREEN, EOM);

if ACTUAL_VALUE = EXPECTED_VALUE then
call WRITE_OUT (@PASS (0»; /* write message to terminal - PASS */
else
call WRITE_OUT (@FAIL (0»; /* write message to terminal - FAIL */
end WRITE_RESULT;

TEST_API_RESPONSE : procedure;

/***************************************************************************/
/* */
/* Performs a test on the API to determine whether it is responding */
/* correctly. */
/* */
办法***************************************************************************/

declare API_VALUE byte;
declare PASS (*) byte data (CR, LF, 'API self-test passed',EOM);
declare FAIL (*) byte data (CR, LF, 'API self-test failed',EOM);

call WRITE_OUT (@WAIT (0»; /* write WAIT message */
call API_SELF_TEST;
if TEST_RESULT.API_BOARD = PASSED then
call WRITE_OUT (@PASS (0»);
else
call WRITE_OUT (@FAIL (0»);
end TEST_API_RESPONSE;
RESET_API : procedure ;

 /******************************************************************************
 * Resets the API card and performs the required handshaking between the
 * primary processor and the API card. If the API responds as expected
 * the test passes, else it fails.
 */
 declare API_VALUE byte;
 declare PASS (*) byte data (CR, LF, 'API ready',EOM);
 declare FAIL (*) byte data (CR, LF, 'API start up fault',EOM);

call WRITE_OUT (@WAIT (0)); /* write WAIT message */

call API_RESET_TEST;
call API_SELF_TEST;
if TEST_RESULT.API_BOARD = PASSED then
  call WRITE_OUT (@PASS (0));
else
  call WRITE_OUT (@FAIL (0));
end RESET_API;
AUDIBLE_ALARM : procedure;

/******************************************************************************
 * This procedure sends a message requesting the number of beeps to be
 * sounded by the audible alarm. When this is entered, it is sent to the
 * dual port RAM of the API. The API then interprets this code, and issues
 * the required acknowledgement. An escape code exits the procedure.
 * 
******************************************************************************/

declare API_VALUE byte;
declare RX_CHAR byte;
declare TEST_END byte;
declare AUDIBLE_ALARM_MESS (*) byte data
  (DIS_HME, STAT_AREA,
   'Select the number of beeps required (0 to F)', CR, LF,
   '(Press ESC to quit)', EOM);

TEST_END = FALSE;
call WRITE_OUT (@AUDIBLE_ALARM_MESS (0));

do while TEST_END = FALSE;
  if ESCAPE = FALSE then
    do;
      RX_CHAR = KEYBOARD_INPUT;
      if (RX_CHAR >= '0') and (RX_CHAR <= '9') then
        do;
          RX_CHAR = RX_CHAR + 30h; /* convert to hex code */
          call WRITE_DPR (@API_INTERRUPT_LOCATION, RX_CHAR);
          API_VALUE = READ_DPR;
          call WRITE_RESULT (API_VALUE, API_READ_ACKNOWLEDGE);
          call WRITE_OUT (@REP_ESC (0));
        end;
      else
        do;
          if (RX_CHAR <= 'F') and (RX_CHAR >= 'A') then
            do;
              RX_CHAR = RX_CHAR + 29h; /* convert to hex code */
              call WRITE_DPR (@API_INTERRUPT_LOCATION, RX_CHAR);
              API_VALUE = READ_DPR;
              call WRITE_RESULT (API_VALUE, API_READ_ACKNOWLEDGE);
              call WRITE_OUT (@REP_ESC (0));
            end;
          else
            do;
              RX_CHAR = RX_CHAR + 30h; /* convert to hex code */
              call WRITE_DPR (@API_INTERRUPT_LOCATION, RX_CHAR);
              API_VALUE = READ_DPR;
              call WRITE_RESULT (API_VALUE, API_READ_ACKNOWLEDGE);
              call WRITE_OUT (@REP_ESC (0));
           end;
        end;
    end;
  else
    do;
      RX_CHAR = KEYBOARD_INPUT;
      if (RX_CHAR >= '0') and (RX_CHAR <= '9') then
        do;
          RX_CHAR = RX_CHAR + 30h; /* convert to hex code */
          call WRITE_DPR (@API_INTERRUPT_LOCATION, RX_CHAR);
          API_VALUE = READ_DPR;
          call WRITE_RESULT (API_VALUE, API_READ_ACKNOWLEDGE);
          call WRITE_OUT (@REP_ESC (0));
        end;
      else
        do;
          if (RX_CHAR <= 'F') and (RX_CHAR >= 'A') then
            do;
              RX_CHAR = RX_CHAR + 29h; /* convert to hex code */
              call WRITE_DPR (@API_INTERRUPT_LOCATION, RX_CHAR);
              API_VALUE = READ_DPR;
              call WRITE_RESULT (API_VALUE, API_READ_ACKNOWLEDGE);
              call WRITE_OUT (@REP_ESC (0));
           end;
          else
            do;
              RX_CHAR = RX_CHAR + 30h; /* convert to hex code */
              call WRITE_DPR (@API_INTERRUPT_LOCATION, RX_CHAR);
              API_VALUE = READ_DPR;
              call WRITE_RESULT (API_VALUE, API_READ_ACKNOWLEDGE);
              call WRITE_OUT (@REP_ESC (0));
           end;
        end;
    end;
  end;
end;
else

    do;
    if (RX_CHAR <= 'f') and (RX_CHAR >= 'a') then
        do;
        RX_CHAR = RX_CHAR + 09h; /* convert to hex code */
        call WRITE_DPR (@API_INTERRUPT_LOCATION, RX_CHAR);
        API_VALUE = READ_DPR;
        call WRITE_RESULT (API_VALUE, API_READ_ACKNOWLEDGE);
        call WRITE_OUT (@REP_ESC (0));
    end;
    else
        TEST_END = FALSE;
    end;
end;
else
    TEST_END = TRUE; /* ESC received */
end;

end AUDIBLE_ALARM;
ASSIGN_VALUES : procedure (CHANNEL);

/******************************************************************************/
  
  * ASSIGN_VALUES assigns values required by the softkey and qwerty keyboard  
  * modules, depending on the channel selected.  
  *  
  ****************************************************************************/

declare CHANNEL byte;
declare ASSIGN byte;

declare SKM1_SUBMENU (*) byte data
  (DIS_CLR, 'Softkey Module 1 Submenu', CR, LF, EOM);

declare SKM2_SUBMENU (*) byte data
  (DIS_CLR, 'Softkey Module 2 Submenu', CR, LF, EOM);

declare SKM3_SUBMENU (*) byte data
  (DIS_CLR, 'Softkey Module 3 Submenu', CR, LF, EOM);

declare QKBM_SUB (*) byte data
  (DIS_CLR,
   'Qwerty keyboard module demonstration menu', CR, LF,
   'A Reset QKBM', CR, LF,
   'B Mask QKBM', CR, LF,
   'C Unmask QKBM', CR, LF,
   'D Enable module stuck key detect', CR, LF,
   'E Disable module stuck key detect', CR, LF,
   'F Mask a bank', CR, LF,
   'G Unmask a bank', CR, LF,
   'H Enable bank stuck key detect', CR, LF,
   'I Disable bank stuck key detect', CR, LF, EOM);
ASSIGN = CHANNEL - '1';

do case ASSIGN;

   do; /* 0 = SKM1 */
      MENU_PTR = &SKM1_SUBMENU;
      MIDS_RESET_CODE = 51h;
      CHANNEL_PASS_CODE = 31h;
      BASE_DPR = &SKM1_CONT.SKM (0);
      DPR_CONTROL = &SKM1_CONT.SKM (1);
      BASE_CODE = SKM1_BASE;
   end;

   do; /* 1 = SKM2 */
      MENU_PTR = &SKM2_SUBMENU;
      MIDS_RESET_CODE = 52h;
      CHANNEL_PASS_CODE = 32h;
      BASE_DPR = &SKM2_CONT.SKM (0);
      DPR_CONTROL = &SKM2_CONT.SKM (1);
      BASE_CODE = SKM2_BASE;
   end;

   do; /* 2 = SKM3 */
      MENU_PTR = &SKM3_SUBMENU;
      MIDS_RESET_CODE = 53h;
      CHANNEL_PASS_CODE = 33h;
      BASE_DPR = &SKM3_CONT.SKM (0);
      DPR_CONTROL = &SKM3_CONT.SKM (1);
      BASE_CODE = SKM3_BASE;
   end;

   do; /* 3 = QKBM */
      MENU_PTR = &QKBM_SUB;
      MIDS_RESET_CODE = 54h;
      CHANNEL_PASS_CODE = 34h;
      BASE_DPR = &QKBM_CONT.QKBM (0);
      DPR_CONTROL = &QKBM_CONT.QKBM (1);
      BASE_CODE = QKBM_BASE;
   end;

   do; /* 4 = EMAC */
      BASE_DPR = &EMAC_CONT.EMAC (0);
      DPR_CONTROL = &EMAC_CONT.EMAC (1);
      BASE_CODE = EMAC_BASE;
   end;

end /* case */

end ASSIGN_VALUES;
RESET_MIDS : procedure;

/**********************************************************
 * A MIDS reset code is sent to the API dual port RAM. The code then waits
 * for the API to reset the manual input devices, and to report the result
 * of its actions. The success/failure of the reset is then sent to the
 * terminal.
 *
/***********************************************************/

declare API_VALUE byte;

call WRITE_DPR (@API_INTERRUPT_LOCATION, MIDS_RESET_CODE);
API_VALUE = READ_DPR;

do while API_VALUE = API_READ_ACKNOWLEDGE;
   API_VALUE = READ_DPR;
end;

call WRITE_RESULT (API_VALUE, CHANNEL_PASS_CODE);
end RESET_MIDS;

SKM_CONTROL : procedure (CONTROL_VALUE);

/**********************************************************
 * This procedure writes a control value to the dual port RAM buffer, the
 * base to the interrupt location and displays the result.
 *
/***********************************************************/

declare CONTROL_VALUE byte;
declare SKM_VALUE_READ byte;

call WRITE_DPR (BASE_DPR, 1);
call WRITE_DPR (DPR_CONTROL, CONTROL_VALUE);
call WRITE_DPR (@API_INTERRUPT_LOCATION, BASE_CODE);
SKM_VALUE_READ = READ_DPR;
call WRITE_RESULT (SKM_VALUE_READ, API_READ_ACKNOWLEDGE);
end SKM_CONTROL;
**In this procedure, the user selects a key for the function required.**

* Up to eight keys are selectable.

---

```plaintext
DECLARE KEY_SEL BYTE;
DECLARE KEY_MESS (*) BYTE DATA
(DIS_HME, STAT_AREA,
 'Select key ', CR, LF,
 '(Press ESC to quit)', EOM);

call WRITE_OUT (GKEY_MESS (0));

IF (ESCAPE = FALSE) AND
(KEYBOARD_INPUT >= '0') AND (KEYBOARD_INPUT <= '7') THEN
do case KEY_SEL;
    call SKM_CONTROL (KEYBOARD_INPUT - '0'); /* mask key */
    call SKM_CONTROL (KEYBOARD_INPUT - '0' + 10h); /* unmask key */
    call SKM_CONTROL (KEYBOARD_INPUT - '0' + 40h); /* enable stk key det */
    call SKM_CONTROL (KEYBOARD_INPUT - '0' + 50h); /* disable stk key det */
end;

end SKM_KEY;
```
SKM_TEST : procedure (SKM_NUMBER);

/************************************************************
 * This is a generic procedure for all the softkey modules. Depending on the *
 * requirement, values are assigned and test or control is performed. *
 ************************************************************/

declare SKM_NUMBER byte;
declare RX_CHAR byte;
declare TEST_END byte;

declare SKM_SUB (*) byte data (DIS_CLR,
    'Softkey module demonstration menu', CR, LF,
    'A Reset SKM', CR, LF,
    'B Mask SKM', CR, LF,
    'C Unmask SKM', CR, LF,
    'D Enable module stuck key detect', CR, LF,
    'E Disable module stuck key detect', CR, LF,
    'F Mask a key', CR, LF,
    'G Unmask a key', CR, LF,
    'H Enable stuck key detect', CR, LF,
    'I Disable stuck key detect', CR, LF, EOM);

call ASSIGN_VALUES (SKM_NUMBER);
call WRITE_OUT (MENU_PTR);
call WRITE_OUT (GSKM_SUB (0));
TEST_END = FALSE;
do while TEST_END = FALSE;
  if ESCAPE = FALSE then
    do;
      call WRITE_OUT (@TEST_MESS (0));
      RX_CHAR = KEYBOARD_INPUT;
      if (RX_CHAR >= 'A') and (RX_CHAR <= 'I') then
        do;
          do case RX_CHAR - 'A';
          call RESET_MIDS; /* 'A' */
          call SKM_CONTROL (MASK_DEVICE); /* 'B' */
          call SKM_CONTROL (UNMASK_DEVICE); /* 'C' */
          call SKM_CONTROL (STUCK_ENABLE); /* 'D' */
          call SKM_CONTROL (STUCK_DISABLE); /* 'E' */
          call SKM_KEY (RX_CHAR - '5'); /* 'F' */
          call SKM_KEY (RX_CHAR - '5'); /* 'G' */
          call SKM_KEY (RX_CHAR - '5'); /* 'H' */
          call SKM_KEY (RX_CHAR - '5'); /* 'I' */
        end; /* case */
      end; /* if (RX_CHAR >= 'A') and (RX_CHAR <= 'I') */
    call WRITE_OUT (@REP_ESC (0));
    TEST_END = FALSE;
  end;
else
  TEST_END = TRUE;
end;
QKBM_BANK: procedure (BANK_SELECTION);

/****************************************************************************
****************************************************************************
* This procedure enables the user to select one of four banks on the qwerty *
* keyboard for the function required. *
****************************************************************************/

declare BANK_SELECTION byte;
declare QKBM_MESS (*) byte data
  (DIS_HME, STAT_AREA, 'Select Bank ', CR, LF, '(Press ESC to quit)', EOM);

call WRITE_OUT (QKBM_MESS (0));
if (ESCAPE = FALSE) and
  (KEYBOARD_INPUT >= '0') and (KEYBOARD_INPUT <= '3') then
do;
do case BANK_SELECTION;
call SKM_CONTROL
  (KEYBOARD_INPUT - '0'); /* mask bank */
call SKM_CONTROL
  (KEYBOARD_INPUT - '0' + 10h); /* unmask bank */
call SKM_CONTROL
  (KEYBOARD_INPUT - '0' + 40h); /* enable stuck bank detection */
call SKM_CONTROL
  (KEYBOARD_INPUT - '0' + 50h); /* disable stuck bank detection */
end; /* case */
end;
end QKBM_BANK;
QKBM_TEST : procedure;

/**************************************************************************

* This is the off-line diagnostic test for the functionality of the qwerty
* keyboard module. Depending on the testing requirement, values are assigned
* and test or control is performed on the selected bank.
*
**************************************************************************/

declare QKBM_NUM byte;
declare RX_CHAR byte;
declare TEST_END byte;

call ASSIGN_VALUES ('4');
call WRITE_OUT (MENU_PTR);
TEST_END = FALSE;

do while TEST_END = FALSE;
    if ESCAPE = FALSE then
do;
        call WRITE_OUT (~TESTMESS (0));
call SEND_CHARACTER (KEYBOARD_INPUT); /* echo operator input */
RX_CHAR = KEYBOARD_INPUT;
        if (RX_CHAR >= 'A') and (RX_CHAR <= 'I') then
do;
            do case RX_CHAR - 'A';
                call RESET_MIDS; /* 'A' */
call SKM_CONTROL (MASK_DEVICE); /* 'B' */
call SKM_CONTROL (UNMASK_DEVICE); /* 'C' */
call SKM_CONTROL (STUCK_ENABLE); /* 'D' */
call SKM_CONTROL (STUCK_DISABLE); /* 'E' */
call QKBM_BANK (RX_CHAR - '5'); /* 'F' */
call QKBM_BANK (RX_CHAR - '5'); /* 'G' */
call QKBM_BANK (RX_CHAR - '5'); /* 'H' */
call QKBM_BANK (RX_CHAR - '5'); /* 'I' */
end; /* case */
        end; /* if (RX_CHAR >= 'A') and (RX_CHAR <= 'I') */
    end;
call WRITE_OUT (GREP_ESC (0));
TEST_END = FALSE;
end;
else
    TEST_END = TRUE;
end;
end QKBM_TEST;
WRITE_STATUS : procedure (STATUS);

 /******************************************************************************
 * WRITE_STATUS writes out one of five possible statuses to the terminal.
 * 
 ******************************************************************************/

 declare STATUS byte;
 declare CONNECTED (*) byte data (' Connected', EOM);
 declare NOT_CONNECTED (*) byte data (' Not connected', EOM);
 declare ST_MESS (*) byte data (' Failed self-test', EOM);
 declare SW_MESS (*) byte data (' Switched', EOM);
 declare FLT (*) byte data (' Fault', EOM);
 declare WRONG_MESS (*) byte data (' Improper code read', EOM);

 if STATUS < 5 then
   do;
     do case STATUS;
       call WRITE_OUT (@CONNECTED (0));
       call WRITE_OUT (@NOT_CONNECTED (0));
       call WRITE_OUT (@FLT (0));
       call WRITE_OUT (@ST_MESS (0));
       call WRITE_OUT (@SW_MESS (0));
     end;
   end;
 else call WRITE_OUT (@WRONG_MESS (0));

 end WRITE_STATUS;
DISPLAY_STATUS : procedure (CHANNEL);

;******************************************************************************************

* DISPLAY_STATUS reads the channel status location of dual port RAM and
* displays the appropriate status of the manual input devices.
*
******************************************************************************************

declare CHANNEL byte;
declare API_VALUE byte;
declare AP_STAT (*) byte data (CR, LF, 'AP 1 - ', EOM);
declare API_STAT (*) byte data (CR, LF, 'API 1 - ', EOM);
declare REM_AP_STAT (*) byte data (CR, LF, 'AP 2 - ', EOM);
declare REM_API_STAT (*) byte data (CR, LF, 'API 2 - ', EOM);
declare EMAC_STAT (*) byte data (CR, LF, 'API Channel 1 - EMAC ', EOM);
declare SKM1_STAT (*) byte data (CR, LF, 'API Channel 2 - SKM ', EOM);
declare SKM2_STAT (*) byte data (CR, LF, 'API Channel 3 - SKM ', EOM);
declare SKM3_STAT (*) byte data (CR, LF, 'API Channel 4 - SKM ', EOM);
declare QKBM_STAT (*) byte data (CR, LF, 'API Channel 5 - QKBM ', EOM);
declare UNKN_STAT (*) byte data (CR, LF, 'API Channel 6 - Spare ', EOM);
declare RBM_STAT (*) byte data (CR, LF, 'API Channel 7 - RBM ', EOM);

do case CHANNEL;
call WRITE_OUT (@AP_STAT (0)); /* 0 */
call WRITE_OUT (@API_STAT (0)); /* 1 */
call WRITE_OUT (@EMAC_STAT (0)); /* 2 */
call WRITE_OUT (@SKM1_STAT (0)); /* 3 */
call WRITE_OUT (@SKM2_STAT (0)); /* 4 */
call WRITE_OUT (@SKM3_STAT (0)); /* 5 */
call WRITE_OUT (@QKBM_STAT (0)); /* 6 */
call WRITE_OUT (@UNKN_STAT (0)); /* 7 */
call WRITE_OUT (@REM_AP_STAT (0)); /* 8 */
call WRITE_OUT (@REM_API_STAT (0)); /* 10 */
end; /* case */

API_VALUE = READ_DPR_BUFFER (@EMAC_STAT_LOC.STATUS (CHANNEL + 20));
call WRITE_STATUS (API_VALUE);

end DISPLAY_STATUS;
WRITE_ASCII : procedure (BINARY_WORD);

/*------------------------------------------------------------------------
 * WRITE_ASCII converts a binary integer value to ASCII and sends the value
 * to the terminal.
 *------------------------------------------------------------------------*/

declare BINARY_WORD word;
declare CHARACTER ARRAY (5) byte;

if (BINARY_WORD and 8000h) = 8000h then
  do;
    call SEND_CHARACTER (2Dh); /* minus sign */
    BINARY_WORD = (not BINARY_WORD);
  end;
else
  call SEND_CHARACTER (2Bh); /* plus sign */

call C030_BIN_WORD_TO_ASCII_DEC (BINARY_WORD, CHARACTER ARRAY);
CHARACTER ARRAY (5) = EOM;
call WRITE_POLL (VDU, @CHARACTER ARRAY);
end WRITE_ASCII;
QKBM_STAT_DISP : procedure;

 /*****************************************************************************
 * Displays the qwerty keyboard key status. *
 */
 /*****************************************************************************

 declare API_VALUE byte;
 declare QKBM_ESC (*) byte data ('QKBM Esc and ', EOM);
 declare QKBM_STAT (*) byte data ('QKBM Key Pressed - ', EOM);

 call WRITE_OUT (@NEXT_LINE (0));
 API_VALUE = READ_DPR_BUFFER (@QKBM_STAT_LOC.STATUS (0));

 if (API_VALUE = 1) or (API_VALUE = 2) then
do;
   if API_VALUE = 2 then
     call WRITE_OUT (@QKBM_ESC (0));
     API_VALUE = READ_DPR_BUFFER (@QKBM_STAT_LOC.STATUS (1));
     call WRITE_OUT (@QKBM_STAT (0));
     call SEND_CHARACTER (API_VALUE);
   end;

 call WRITE_DPR (@API_INTERRUPT_LOCATION, AP_READ_ACKNOWLEDGE);

 end QKBM_STAT_DISP;
KEY_STAT_DISP : procedure;

avigation Functions

宣言API_VALUE byte;
declare KEY_GROUP byte;
declar STATUS_READ byte;
declar SKM1_STATUS (*) byte data ('SKM1 - ', EOM);
declar SKM2_STATUS (*) byte data ('SKM2 - ', EOM);
declar SKM3_STATUS (*) byte data ('SKM3 - ', EOM);
declar QKBH_STATUS (*) byte data ('QKBH - ', EOM);
declar STK_DET (*) byte data ('Stuck Key Detected = ', EOM);
declar STK_CLR (*) byte data ('Stuck Key Cleared = ', EOM);

API_VALUE = READ_DPR_BUFFER (GKEY_STAT_LOC.STATUS (0));
if API_VALUE = 1 then
do;
API_VALUE = READ_DPR_BUFFER (GKEY_STAT_LOC.STATUS (1));
KEY_GROUP = shr (API_VALUE, 4) and 0fh;
if KEY_GROUP < 0Ah then
do;
call WRITE_OUT (@NEXT_LINE (0));
do case KEY_GROUP;
do; /* 0 - skm1 */
call WRITE_OUT (@SKM1_STATUS (0));
call WRITE_OUT (@STK_DET (0));
end;
do; /* 1 - skm1 */
call WRITE_OUT (@SKM1_STATUS (0));
call WRITE_OUT (@STK_CLR (0));
end;
do; /* 2 - skm2 */
call WRITE_OUT (@SKM2_STATUS (0));
call WRITE_OUT (@STK_DET (0));
end;
do; /* 3 - skm2 */
call WRITE_OUT (@SKM2_STATUS (0));
call WRITE_OUT (@STK_CLR (0));
end;
do; /* 4 - skm3 */
call WRITE_OUT (@SKM3_STATUS (0));
call WRITE_OUT (@STK_DET (0));
end;
do; /* 5 - skm3 */
call WRITE_OUT (@SKM3_STATUS (0));
call WRITE_OUT (@STK_CLR (0));
end;
do; /* 6 - rbm */
end;
do; /* 7 - rbm */
end;
do; /* 8 - qkbm */
call WRITE_OUT (@QKBM_STATUS (0));
call WRITE_OUT (@STK_DET (0));
end;
do; /* 9 - qkbm */
call WRITE_OUT (@QKBM_STATUS (0));
call WRITE_OUT (@STK_CLR (0));
end;
end; /* case */

STATUS_READ = (API_VALUE and 0fh) or 30h;
call SEND_CHARACTER (STATUS_READ);
end;
end;
call WRITE_DPR (@API_INTERRUPT_LOCATION, AP_READ_ACKNOWLEDGE);
end KEY_STAT_DISP;
RBM_STAT_DISP : procedure;

 /******************************************************************************
 * An update of the rollerball module value is to be kept which is added or
 * subtracted when data is processed. On the terminal the values are updated
 * on a single line.
 */
*******************************************************************************/

declare API_VALUE byte;
declare RBM_X (*) byte data (STAT_AREA, 'RBn X = ', EOM);
declare RBM_Y (*) byte data (STAT1_AREA, 'RBn Y = ', EOM);
declare RBM_KEY (*) byte data (STAT2_AREA, 'RBn Key Pressed ', EOM);

API_VALUE = READ_DPR_BUFFER (@RBM_STAT_LOC.STATUS (0));
if API_VALUE = 3 then
do;
   API_VALUE = READ_DPR_BUFFER (@RBM_STAT_LOC.STATUS (1));
   if (API_VALUE and 80h) = 80h then
      RBM_X_VAL = RBM_X_VAL - (API_VALUE and 7fh);
   else
      RBM_X_VAL = RBM_X_VAL + (API_VALUE and 7fh);
   call WRITE_OUT (@RBM_X (0));
call WRITE_ASCII (RBM_X_VAL);
   API_VALUE = READ_DPR_BUFFER (@RBM_STAT_LOC.STATUS (2));
   if (API_VALUE and 80h) = 80h then
      RBM_Y_VAL = RBM_Y_VAL - (API_VALUE and 7fh);
   else
      RBM_Y_VAL = RBM_Y_VAL + (API_VALUE and 7fh);
   call WRITE_OUT (@RBM_Y (0));
call WRITE_ASCII (RBM_Y_VAL);
   API_VALUE = READ_DPR_BUFFER (@RBM_STAT_LOC.STATUS (3));
if (API_VALUE > 0) and (API_VALUE <= 7) then
  do;
    call WRITE_OUT (@RBM_KEY (0));
    if (API_VALUE and 1h) = 1h then
      call SEND_CHARACTER ('1');
    call SEND_CHARACTER (' '); 
    call SEND_CHARACTER (' '); 
    if (API_VALUE and 2h) = 2h then
      call SEND_CHARACTER ('2');
    call SEND_CHARACTER (' '); 
    call SEND_CHARACTER (' '); 
    if (API_VALUE and 4h) = 4h then
      call SEND_CHARACTER ('3');
    end;
  end;
end;
call WRITE_DPR (GAPI_INTERRUPT_LOCATION, AP_READ_ACKNOWLEDGE);
end RBM_STAT_DISP;

SPARE_STAT_DISP : procedure;

/*****************************************************************************/
* * 
* Displays the status of the spare channel. 
* *
/*****************************************************************************/

declare API_VALUE byte;
declare SPARE_STATUS (*) byte data ('Spare Channel Data = ', EOM);

API_VALUE = READ_DPR_BUFFER (GSPARE_STAT_LOC.STATUS (0));

if API_VALUE = 1 then
  do;
    API_VALUE = READ_DPR_BUFFER (GKEY_STAT_LOC.STATUS (1));
    call WRITE_OUT (@NEXT_LINE (0));
    call WRITE_OUT (@SPARE_STATUS (0));
    call SEND_CHARACTER (API_VALUE);
  end;

call WRITE_DPR (GAPI_INTERRUPT_LOCATION, AP_READ_ACKNOWLEDGE);
end SPARE_STAT_DISP;
EMAC_ALL_STATUS : procedure (TYPE_STATUS);

declare TYPE_STATUS byte;
declare VALUE_READ byte;
declare TEMP_READ word;
declare I byte;
declare TEMP_MESS (* byte data (CR, LF, 'Temperature (Kelvin) = ', EOM);
declare HUMID_MESS (* byte data (CR, LF, 'Humidity = ', EOM);
declare MODE_MESS (* byte data (CR, LF, 'Mode = ', EOM);
declare ACH_MESS (* byte data (CR, LF, 'Self Test = ', EOM);
declare BIT_MESS (* byte data (CR, LF, 'Heater = ', EOM);
declare PS5_MESS (* byte data (CR, LF, 'GPSU 1 5V Supply = ', EOM);
declare PS12_MESS (* byte data (CR, LF, 'GPSU 1 12V Supply = ', EOM);
declare PS2_5_MESS (* byte data (CR, LF, 'GPSU 2 5V Supply = ', EOM);
declare PS2_12_MESS (* byte data (CR, LF, 'GPSU 2 12V Supply = ', EOM);
declare MAINS_MESS (* byte data (CR, LF, 'AC Mains Supply = ', EOM);
declare FREQ_MESS (* byte data (CR, LF, 'Frequency = ', EOM);
declare AUXS_MESS (* byte data (CR, LF, 'AUXS Supply = ', EOM);
declare GDU1_MESS (* byte data (CR, LF, 'GDU 1 Interlock = ', EOM);
declare GDU2_MESS (* byte data (CR, LF, 'GDU 2 Interlock = ', EOM);
declare GDU3_MESS (* byte data (CR, LF, 'GDU 3 Interlock = ', EOM);
declare APCC_MESS (* byte data (CR, LF, 'APCC 1 Interlock = ', EOM);
declare APCC2_MESS (* byte data (CR, LF, 'APCC 2 Interlock = ', EOM);
declare FAHD_MESS (* byte data (CR, LF, 'FAHD Interlock = ', EOM);
declare CSPM1_MESS (* byte data (CR, LF, 'CSPM Bank 1 Switch = ', EOM);
declare CSPM2_MESS (* byte data (CR, LF, 'CSPM Bank 2 Switch = ', EOM);
declare VENT_MESS (* byte data (CR, LF, 'Ventilation Voltage = ', EOM);
declare AP2_MESS (* byte data (CR, LF, 'AP 2 = ', EOM);
declare API2_MESS (* byte data (CR, LF, 'API 2 Self Test = ', EOM);
declare EMAC_PASS (* byte data ('Passed', EOM);
declare EMAC_FAIL (* byte data ('Failed', EOM);
declare EMAC_MAST (* byte data ('Master', EOM);
declare EMAC_SLAV (* byte data ('Slave', EOM);
declare EMAC_TEST (* byte data ('Test', EOM);
declare EMAC_ON (* byte data ('On', EOM);
declare EMAC_OFF (* byte data ('Off', EOM);
declare EMAC_OVER (* byte data ('Over', EOM);
declare EMAC_UNDER (* byte data ('Under', EOM);
declare EMAC_CLOSED (* byte data ('Closed', EOM);
declare EMAC_OPEN (* byte data ('Open', EOM);
do case TYPE_STATUS - 1;

  do; /* 1 = temperature status */
  VALUE_READ = READ_DPR_BUFFER (EMAC_STAT_LOC.STATUS (1));
  call WRITE_OUT (@TEMP_MESS (0));
  TEMP_READ = double (VALUE_READ) + 248;
  call WRITE_ASCII (TEMP_READ);
  end;

  do; /* 2 = humidity */
  VALUE_READ = READ_DPR_BUFFER (EMAC_STAT_LOC.STATUS (2));
  call WRITE_OUT (@HUMID_MESS (0));
  if VALUE_READ = 0 then call WRITE_OUT (@EMAC_PASS (0));
  else call WRITE_OUT (@EMAC_FAIL (0));
  end;

  do; /* 3 = emac mode */
  VALUE_READ = READ_DPR_BUFFER (EMAC_STAT_LOC.STATUS (3));
  call WRITE_OUT (@MODE_MESS (0));
  if VALUE_READ < 4 then
    do;
    do case VALUE_READ;
      call WRITE_OUT (@EMAC_MAST (0));
      call WRITE_OUT (@EMAC_SLAV (0));
      call WRITE_OUT (@EMAC_TEST (0));
    ; /* reserved */
    end;
    end;
  else
    call WRITE_OUT (@EMAC_FAIL (0));
  end;

  do; /* 4 = emac self test */
  VALUE_READ = READ_DPR_BUFFER (EMAC_STAT_LOC.STATUS (5));
  call WRITE_OUT (@BIT_MESS (0));
  if VALUE_READ = 0 then
    call WRITE_OUT (@EMAC_PASS (0));
  else
    call WRITE_OUT (@EMAC_FAIL (0));
  end;

  do; /* 5 = anti condensation heater */
  VALUE_READ = READ_DPR_BUFFER (EMAC_STAT_LOC.STATUS (4));
  call WRITE_OUT (@ACH_MESS (0));
  if VALUE_READ = 0 then
    call WRITE_OUT (@EMAC_ON (0));
  else
    call WRITE_OUT (@EMAC_OFF (0));
  end;
do; /* 6 = 5v power */
VALUE_READ = READ_DPR_BUFFER @EMAC_STAT_LOC.STATUS (6);
call WRITE_OUT @PS5_MESS (0);
if VALUE_READ < 3 then
do;
docase VALUE_READ;
call WRITE_OUT @EMAC_ON (0);
call WRITE_OUT @EMAC_OVER (0);
call WRITE_OUT @EMAC_UNDER (0);
; /* reserved */
end;
end;
else
call WRITE_OUT @EMAC_FAIL (0);
end;

do; /* 7 = 12v power */
VALUE_READ = READ_DPR_BUFFER @EMAC_STAT_LOC.STATUS (7);
call WRITE_OUT @PS12_MESS (0);
if VALUE_READ < 4 then
do;
docase VALUE_READ;
call WRITE_OUT @EMAC_ON (0);
call WRITE_OUT @EMAC_OVER (0);
call WRITE_OUT @EMAC_UNDER (0);
; /* reserved */
end;
end;
else
call WRITE_OUT @EMAC_FAIL (0);
end;

do; /* 8 = mains */
VALUE_READ = READ_DPR_BUFFER @EMAC_STAT_LOC.STATUS (8);
call WRITE_OUT @MAINS_MESS (0);
if VALUE_READ < 4 then
do;
docase VALUE_READ;
call WRITE_OUT @EMAC_ON (0);
call WRITE_OUT @EMAC_OVER (0);
call WRITE_OUT @EMAC_UNDER (0);
; /* reserved */
end;
end;
else
call WRITE_OUT @EMAC_FAIL (0);
end;
do; /* 9 = frequency */
VALUE_READ = READ_DPR_BUFFER (GEMAC_STAT_LOC.STATUS (9));
call WRITE_OUT (GREQ_MESS (0));
if VALUE_READ = 0 then
  call WRITE_OUT (GEMAC_PASS (0));
else
  call WRITE_OUT (GEMAC_FAIL (0));
end;

do; /* 10 = auxiliary */
VALUE_READ = READ_DPR_BUFFER (GEMAC_STAT_LOC.STATUS (10));
call WRITE_OUT (GAUXS_MESS (0));
if VALUE_READ = 0 then
  call WRITE_OUT (GEMAC_PASS (0));
else
  call WRITE_OUT (GEMAC_FAIL (0));
end;

do; /* 11 = graphics display unit 1 interlock */
VALUE_READ = READ_DPR_BUFFER (GEMAC_STAT_LOC.STATUS (11));
call WRITE_OUT (GDU1_MESS (0));
if VALUE_READ = 0 then
  call WRITE_OUT (GEMAC_PASS (0));
else
  call WRITE_OUT (GEMAC_FAIL (0));
end;

do; /* 12 = graphics display unit 2 interlock */
VALUE_READ = READ_DPR_BUFFER (GEMAC_STAT_LOC.STATUS (12));
call WRITE_OUT (GDU2_MESS (0));
if VALUE_READ = 0 then
  call WRITE_OUT (GEMAC_PASS (0));
else
  call WRITE_OUT (GEMAC_FAIL (0));
end;

do; /* 13 = graphics display unit 3 interlock */
VALUE_READ = READ_DPR_BUFFER (GEMAC_STAT_LOC.STATUS (13));
call WRITE_OUT (GDU3_MESS (0));
if VALUE_READ = 0 then
  call WRITE_OUT (GEMAC_PASS (0));
else
  call WRITE_OUT (GEMAC_FAIL (0));
end;
do; /* 14 = applications processor card cage 1 interlock */
VALUE_READ = READ_DPR_BUFFER (GEMAC_STAT_LOC.STATUS (14));
call WRITE_OUT (@APCC1_MESS (0));
if VALUE_READ = 0 then
    call WRITE_OUT (@GEMAC_PASS (0));
else
    call WRITE_OUT (@GEMAC_FAIL (0));
end;

do; /* 15 = applications processor card cage 2 interlock */
VALUE_READ = READ_DPR_BUFFER (GEMAC_STAT_LOC.STATUS (15));
call WRITE_OUT (@APCC2_MESS (0));
if VALUE_READ = 0 then
    call WRITE_OUT (@GEMAC_PASS (0));
else
    call WRITE_OUT (@GEMAC_FAIL (0));
end;

do; /* 16 = fan housing drawer interlock */
VALUE_READ = READ_DPR_BUFFER (GEMAC_STAT_LOC.STATUS (16));
call WRITE_OUT (@FAHD_MESS (0));
if VALUE_READ = 0 then
    call WRITE_OUT (@GEMAC_PASS (0));
else
    call WRITE_OUT (@GEMAC_FAIL (0));
end;

do; /* 17 = all control and status panel module bank 1 switches */
VALUE_READ = READ_DPR_BUFFER (GEMAC_STAT_LOC.STATUS (17));
do I = 0 to 7;
call TIME (20);
call WRITE_OUT (@CSPM1_MESS (0));
call SEND_CHARACTER (I + 30h);
if (shr (VALUE_READ, I) and 01h) = 0 then
    call WRITE_OUT (@GEMAC_CLOSED (0));
else
    call WRITE_OUT (@GEMAC_OPEN (0));
end;
end;
do; /* 18 = all control and status panel module bank 2 switches */
VALUE_READ = READ_DPR_BUFFER (GEMAC_STAT_LOC.STATUS (18));
do I = 0 to 7;
call TIME (20);
call WRITE_OUT (QCSPM2_MESS (0));
call SEND_CHARACTER (I + 30h);
if (shr (VALUE_READ, I) and 01h) = 0 then
call WRITE_OUT (GEMAC_CLOSED (0));
else
call WRITE_OUT (GEMAC_OPEN (0));
end;
end;
do; /* 19 = ventilation voltage */
VALUE_READ = READ_DPR_BUFFER (GEMAC_STAT_LOC.STATUS (19));
call WRITE_OUT (GVENT_MESS (0));
call WRITE_ASCI I (12 - (VALUE_READ mod 8));
end;
call DISPLAY_STATUS (0); /* 20 = applications processor 1 */
call DISPLAY_STATUS (1); /* 21 = API 1 */
call DISPLAY_STATUS (2); /* 22 = API ch 1 */
call DISPLAY_STATUS (3); /* 23 = API ch 2 */
call DISPLAY_STATUS (4); /* 24 = API ch 3 */
call DISPLAY_STATUS (5); /* 25 = API ch 4 */
call DISPLAY_STATUS (6); /* 26 = API ch 5 */
call DISPLAY_STATUS (7); /* 27 = API ch 6 */
call DISPLAY_STATUS (8); /* 28 = API ch 7 */
call DISPLAY_STATUS (9); /* 29 = remote applications processor */
call DISPLAY_STATUS (10); /* 30 = remote API */
do; /* 31 = 5v power */
VALUE_READ = READ_DPR_BUFFER (GEMAC_STAT_LOC.STATUS (31));
call WRITE_OUT (GPS2_5_MESS (0));
if VALUE_READ < 4 then 
do;
do case VALUE_READ;
call WRITE_OUT (GEMAC_ON (0));
call WRITE_OUT (GEMAC_OVER (0));
call WRITE_OUT (GEMAC_UNDER (0));
; /* reserved */
end;
end;
else
call WRITE_OUT (GEMAC_FAIL (0));
end;
do; /* 32 = 12v power */
VALUE_READ = READ_DPR_BUFFER (GEMAC_STAT_LOC.STATUS (32));
call WRITE_OUT (GPS2_12_MESS (0));
if VALUE_READ < 4 then
  do case VALUE_READ;
    call WRITE_OUT (GEMAC_ON (0));
    call WRITE_OUT (GEMAC_OVER (0));
    call WRITE_OUT (GEMAC_UNDER (0));
    ; /* reserved */
  end;
else
  call WRITE_OUT (GEMAC_FAIL (0));
end;
end; /* case */
end EMAC_ALL_STATUS;

SWITCH_STAT: procedure;

declare RX_CHAR byte;
declare REQ_MESS (*) byte data
  (DIS_HME, STAT_AREA,
   'Control and status panel module switch menu', CR, LF,
   CR, LF,
   'A Bank 1',
   'B Bank 2',
   CR, LF, EOM);
call WRITE_OUT (GREQ_MESS (0));
if (ESCAPE = FALSE) and
  (KEYBOARD_INPUT >= 'A') and (KEYBOARD_INPUT <= 'B') then
do;
  RX_CHAR = KEYBOARD_INPUT;
call SEND_CHARACTER (KEYBOARD_INPUT);
  do case (RX_CHAR - 'A');
    call SKM_CONTROL (RX_CHAR - 'A' + 30h); /* 30h - 37h */
    call SKM_CONTROL (RX_CHAR - 'A' + 38h); /* 38h - 3fh */
  end;
if READ_DPR = 13h then
  call EMAC_ALL_STATUS (RX_CHAR - 30h + 17);
end;
end SWITCH_STAT;
POWER_SUPPLY_STATUS : procedure;

declare TEST_END byte;
declare RX_CHAR byte;
declare POWER_SUPPLY_SUBMENU (*) byte data
  (DIS_CLR, CR,
   'Power supply menu', CR, LF,
   LF,
   'A' Power supply unit 1 5V', CR, LF,
   'B' Power supply unit 1 12V', CR, LF,
   'C' Mains', CR, LF,
   'D' Frequency', CR, LF,
   'E' Auxiliaries', CR, LF,
   'F' Power supply unit 2 5V', CR, LF,
   'G' Power supply unit 2 12V', CR, LF, ECM);

call WRITE_OUT (@POWER_SUPPLY_SUBMENU (0));
call WRITE_POLL (VDU, @PROMPT);

TEST_END = FALSE;

do while TEST_END = FALSE;
  if ESCAPE = FALSE then
    do;
      call WRITE_OUT (@TEST_MESS (0));
      call SEND_CHARACTER (KEYBOARD_INPUT);
      call WRITE_OUT (@NEXT_LINE (0));
      RX_CHAR = KEYBOARD_INPUT;
      if (RX_CHAR >= 'A') and (RX_CHAR <= 'G') then
        do;
          call SKM_CONTROL (KEYBOARD_INPUT - 31h);
          if READ_DPR = 13h then
            call EMAC_ALL_STATUS (RX_CHAR - 'A' + 6);
        end;
      if (RX_CHAR >= 'F') and (RX_CHAR <= 'G') then
        do;
          call SKM_CONTROL (KEYBOARD_INPUT - 31h);
          if READ_DPR = 13h then
            call EMAC_ALL_STATUS (RX_CHAR - 'A' + 31);
        end;

      TEST_END = FALSE;
    end;
  else
    TEST_END = TRUE;
  end; /* while TEST_END = FALSE */
end POWER_SUPPLY_STATUS;
INTERLOCK_STATUS : procedure;

declare RX_CHAR byte;
declare TEST_END byte;

declare INTERLOCK_STATUS_SUBMENU (*) byte data
(DIS_CLR, CR,
 'Interlock Menu', CR, LF, LF,
 'A Graphics display unit 1', CR, LF,
 'B Graphics display unit 2', CR, LF,
 'C Graphics display unit 3', CR, LF,
 'D Card cage 1', CR, LF,
 'E Card cage 2', CR, LF,
 'F Fan housing drawer', CR, LF, EOM);

call WRITE_OUT (@INTERLOCK_STATUS_SUBMENU (0));
TEST_END = FALSE;
do while TEST_END = FALSE;
  if ESCAPE = FALSE then
    do;
      call WRITE_OUT (@TEST_MESS (0));
      call SEND_CHARACTER (KEYBOARD_INPUT);
      call WRITE_OUT (@NEXT_LINE (0));
      RX_CHAR = KEYBOARD_INPUT;
      if (RX_CHAR >= 'A') and (RX_CHAR <= 'F') then
        do;
          call SKM_CONTROL (KEYBOARD_INPUT - 21h);
          if READ_DPR = 13h then
            call EMAC_ALL_STATUS (RX_CHAR - 'A' + 11);
        end;
      TEST_END = FALSE;
    end;
  else
    TEST_END = TRUE;
  end;

end INTERLOCK_STATUS;
WRITE_MIDS_RESULT : procedure (MIDS_VAL);

/***************************************************************************
 * The result of the received data is displayed on the terminal.   
 *                                                           
 **************************************************************************/

declare MIDS_VAL byte;
declare TEMP_VAL byte;
declare SKM1_RESULT (*) byte data (CR, LF, 'SKM1 Key - ', EOM);
declare SKM2_RESULT (*) byte data (CR, LF, 'SKM2 Key - ', EOM);
declare SKM3_RESULT (*) byte data (CR, LF, 'SKM3 Key - ', EOM);

if (MIDS_VAL >= 80h) and (MIDS_VAL <= 87h) then
do;
call WRITE_OUT (@SKM1_RESULT (0));
TEMP_VAL = MIDS_VAL - 80h;
call SEND_CHARACTER (TEMP_VAL);
end;

if (MIDS_VAL >= 90h) and (MIDS_VAL <= 97h) then
do;
call WRITE_OUT (@SKM2_RESULT (0));
TEMP_VAL = MIDS_VAL - 60h;
call SEND_CHARACTER (TEMP_VAL);
end;

if (MIDS_VAL >= OA0h) and (MIDS_VAL <= OA7h) then
do;
call WRITE_OUT (@SKM3_RESULT (0));
TEMP_VAL = MIDS_VAL - 70h;
call SEND_CHARACTER (TEMP_VAL);
end;

if (MIDS_VAL = 10h) then
call KEY_STAT_DISP;

if (MIDS_VAL = 11h) then
call QKBM_STAT_DISP;

if (MIDS_VAL = 12h) then
call RBM_STAT_DISP;
if (MIDS_VAL = 13h) then
do;
    TEMP_VAL = READ_DPR_BUFFER (GEMAC_STAT_LOC.STATUS (0));
if TEMP_VAL <= 32 then
    call EMAC_ALL_STATUS (TEMP_VAL);
    call WRITE_DPR (@API_INTERRUPT_LOCATION, AP_READ_ACKNOWLEDGE);
end;

if (MIDS_VAL = 14h) then
    call SPARE_STAT_DISP;

end WRITE_MIDS_RESULT;

ALL_MIDS_TEST : procedure;

/******************************************************************************
* 
* This procedure causes any MIDS transmission to be displayed. 
* It is terminated by a keyboard interrupt.
* 
*******************************************************************************/
declare API_VALUE byte;
do API_VALUE = 0 to 8;
    call DISPLAY_STATUS (API_VALUE);
end;
KEYBOARD_INTERRUPT = FALSE;
do while KEYBOARD_INTERRUPT = FALSE;
    if DPR_INTERRUPT = TRUE then
        do;
            call MASK_PIC (API_INTERRUPT_NO);
            API_VALUE = DPR_RX;
            DPR_INTERRUPT = FALSE;
            call WRITE_MIDS_RESULT (API_VALUE);
            call WRITE_OUT (ERASE_NEXT (0));
            call UNMASK_PIC (API_INTERRUPT_NO);
        end;
    end;
KEYBOARD_INTERRUPT = FALSE;
end ALL_MIDS_TEST;
EMAC_STATUS_REQUEST : procedure;

declare TEST_END byte;
declare I byte;
declare RX_CHAR byte;
declare EMAC_STAT_SUBMENU (*) byte data
  (DIS_CLR, CR,
   'Status request menu', CR, LF,
   'A Temperature', CR, LF,
   'B Humidity', CR, LF,
   'C EMAC operating mode', CR, LF,
   'D EMAC self-test', CR, LF,
   'E Anti-condensation heating', CR, LF,
   'F Power supply menu', CR, LF,
   'G Interlocks menu', CR, LF,
   'H Control and status panel menu', CR, LF,
   'I Ventilation voltage', CR, LF, EOM);

call WRITE_OUT (GEMAC_STAT_SUBMENU (0));
call WRITE_POLL (VDU, SPROMPT);

TEST_END = FALSE;
do while TEST_END = FALSE;
  if ESCAPE = FALSE then
    do;
      call WRITE_OUT (GTEST_MESS (0));
      call SEND_CHARACTER (KEYBOARD_INPUT);
      call WRITE_OUT (GNEXT_LINE (0));
      RX_CHAR = KEYBOARD_INPUT;
      if (RX_CHAR >= 'A') and (RX_CHAR <= 'C') then
        do;
          call SKM_CONTROl (KEYBOARD_INPUT - 'A');
          if READ_DPR = 13h then
            call EMAC_ALL_STATUS (RX_CHAR - 'A' + 1);
          call WRITE_OUT (GERASE_NEXT (0));
        end;
if (RX_CHAR >= 'D') and (RX_CHAR <= 'I') then
  do;
  do case RX_CHAR - 'D';
    do; /* RX_CHAR = 'D' */
      call SKM_CONTROL (5);
      if READ_DPR = 13h then
        call EMAC_ALL_STATUS (5);
      end;
      call POWER_SUPPLY_STATUS; /* RX_CHAR = 'E' */
      call INTERLOCK_STATUS; /* RX_CHAR = 'F' */
      call SWITCH_STAT; /* RX_CHAR = 'G' */
    do; /* RX_CHAR = 'H' */
      call SKM_CONTROL (4);
      if READ_DPR = 13h then
        call EMAC_ALL_STATUS (19);
      end;
    do; /* RX_CHAR = 'I' */
      do I = 20 to 30;
        call EMAC_ALL_STATUS (I);
      end;
      TEST_END = FALSE;
    end;
  end; /* case */
end; /* if (RX_CHAR >= 'D') and (RX_CHAR <= 'I') */
else
  TEST_END = TRUE;
end; /* while */
end EMAC_STATUS_REQUEST;
EMAC_CONTROL_REQUEST : procedure;

declare TEST_END byte;
declare I byte;
declare RX_CHAR byte;
declare EMAC_CONT_SUBMENU (*) byte data
(DIS_CLR, CR,
 'Control mode menu', CR, LF,
 LF,
 'A  Set all user lamps on', CR, LF,
 'B  Set all user lamps off', CR, LF,
 'C  Set all user LEDs on', CR, LF,
 'D  Set all user LEDs off', CR, LF, EOM);

call WRITE_OUT (EMAC_CONT_SUBMENU (0));
call WRITE_POLL (VDU, @PROMPT);

TEST_END = FALSE;
do while TEST_END = FALSE;
  if ESCAPE = FALSE then
    do;
      call WRITE_OUT (@TEST_MESS (0));
call SEND_CHARACTER (KEYBOARD_INPUT);
call WRITE_OUT (@NEXT_LINE (0));
RX_CHAR = KEYBOARD_INPUT;
if (RX_CHAR >= 'A') and (RX_CHAR <= 'D') then
  do;
do case RX_CHAR - 'A';

  do; /* User lamps on */
do I = 5 to 0fh;
call SKM_CONTROL (50h + I);
end;
end;

  do; /* User lamps off */
do I = 5 to 0fh;
call SKM_CONTROL (60h + I);
end;
end;
do; /* UserLEDs on */
  do I = 0 to 03h;
    call SKM_CONTROL (70h + I);
  end;
end;

do; /* UserLEDs off */
  do I = 0 to 03h;
    call SKM_CONTROL (74h + I);
  end;
end; /* case */
end; /* if (RX_CHAR >= 'A') and (RX_CHAR <= 'D') */
TEST_END = FALSE;
end; /* if ESCAPE = FALSE */
else
  TEST_END = TRUE;
end; /* while */
end EMAC_CONTROL_REQUEST;

EMAC_DIAGNOSTICS : procedure;

declare TEST_END byte;
declare RX_MIDS byte;
declare I byte;
declare MORE_MESS (*) byte data
  (CR, LF,
   ' ********* Press <ESC> to continue ********', EOM);
declare EMAC_SUBMENU (*) byte data
  (DIS_CLR, CR,
   'Environmental monitoring and control demonstration menu', CR, LF,
    LF,
   'A  Status of console', CR, LF,
   'B  Status request menus', CR, LF,
   'C  Control mode menu', CR, LF,
   'D  Reset EMAC', CR, LF, EOM);

TEST_END = FALSE;
call ASSIGN_VALUES ('5'); /* EMAC */
do while TEST_END = FALSE;
call WRITE_OUT (EMAC_SUBMENU (0));
call WRITE_POLL (VDU, @PROMPT);
if ESCAPE = FALSE then
do;
call WRITE_OUT (@TEST_MESS (0));
call SEND_CHARACTER (KEYBOARD_INPUT);
call WRITE_OUT (@NEXT_LINE (0));
RX_MIDS = KEYBOARD_INPUT;
if (RX_MIDS >= 'A') and (RX_MIDS <= 'D') then
do;
do case RX_MIDS - 'A';
  do; /* A */
do l = 0 to 32;
  if (l = 17) or (l=18) or (l=20) or (l=31) then
do;
call WRITE_OUT (@MORE_MESS (0));
do while not ESCAPE;
  end;
  end;
call EMAC_ALL_STATUS (l);
do while not ESCAPE;
  end;
end;
call EMAC_STATUS_REQUEST; /* B */
call EMAC_CONTROL_REQUEST; /* C */
call SKM_CONTROL (Offh); /* D */
end; /* case */
end; /* if (RX_MIDS >= 'A') and (RX_MIDS <= 'D') */

TEST_END = FALSE;
end; /* if ESCAPE = FALSE */
else
  TEST_END = TRUE;
end;
end EMAC_DIAGNOSTICS;
MIDS_TEST : procedure ;

/*******************************************************************************************/
/* Performs MIDS related tests */
/* */
*************************************************************************************************************/
declare TEST_END byte;
declare RX_MIDS byte;
declare MIDS_SUBMENU (*) byte data
(DIS_CLR, CR,
 'Manual input devices demonstration menu', CR, LF,
 'A SKM 1 menu', CR, LF,
 'B SKM 2 menu', CR, LF,
 'C SKM 3 menu', CR, LF,
 'D QKBM menu', CR, LF,
 'E EMAC menu', CR, LF,
 'F Console functionality test', CR, LF, EOM);

TEST_END = FALSE;
do while TEST_END = FALSE;
call WRITE_OUT (@MIDS_SUBMENU (0»;
call WRITE_POLL (VDU, @PROMPT);
if ESCAPE = FALSE then
do;
call WRITE_OUT (@TEST_MESS (0»;
call SEND_CHARACTER (KEYBOARD_INPUT);
call WRITE_OUT (@NEXT_LINE (0»;
RX_MIDS = KEYBOARD_INPUT;
if (RX_MIDS >= 'A') and (RX_MIDS <= 'F') then
do;
do case RX_MIDS - 'A';
call SKM_TEST (RX_MIDS - 'A' + 30h); /* A */
call SKM_TEST (RX_MIDS - 'A' + 30h); /* B */
call SKM_TEST (RX_MIDS - 'A' + 30h); /* C */
call QKBM_TEST;
/* D */
call EMAC_DIAGNOSTICS; /* E */
call ALL_MIDS_TEST; /* F */
end; /* case */
end; /* if */
call WRITE_OUT (@REP_ESC (0»;
TEST_END = FALSE;
end; /* if ESCAPE = FALSE */
else
  TEST_END = TRUE;
end; /* while */
end MIDS_TEST;
API_TEST : procedure ;

declare TEST_END byte;
declare RX_API byte;
declare API_SUBMENU (*) byte data
  (DIS_CLR, CR,
  'Applications processor interface demonstrations menu', CR, LF,
  'A' 'Reset API and all MIDS', CR, LF,
  'B' Involve API self-test', CR, LF,
  'C' Perform audible alarm test', CR, LF, EOM);

TEST_END = FALSE;
call WRITE_OUT (API_SUBMENU (0));
call WRITE_POLL (VDU, SPROMPT);

do while TEST_END = FALSE;
  if ESCAPE = FALSE then
    do;
    call WRITE_OUT (TEST_MESS (0));
    call SEND_CHARACTER (KEYBOARD_INPUT);
    call WRITE_OUT (GWEXT_LINE (0));
    RX_API = KEYBOARD_INPUT;
    if (RX_API >= 'A') and (RX_API <= 'C') then
      do;
        do case RX_API - 'A';
          call RESET_API; /* A */
          call TEST_API_RESPONSE; /* B */
          call AUDIBLE_ALARM; /* C */
        end; /* case */
      end; /* if */
      TEST_END = FALSE;
    end; /* if ESCAPE = FALSE */
  else
    TEST_END = TRUE;
  end; /* while */
end API_TEST;
CONSOLE_DRIVER: procedure public;

declare CONSOLE_MENU (*) byte data

(CLR_SCREEN,
 'Console demonstration menu', CR, LF,
 'A Application processor interface demonstration', CR, LF,
 'B Manual input devices demonstration', CR, LF, LF, EOM);

declare LETTER byte;

disable; /* disable interrupts */
call INITIALISE_VARIABLES;
call INITIALISE_INTERRUPT_VECTORS;
enable; /* enable interrupts */

call WRITE_OUT (@CONSOLE_MENU);
call WRITE_OUT (@PROMPT);
call RESET_API;

LETTER = INPUT_CHARACTER (KB);

do while LETTER <> ESC;
  LETTER = INPUT_CHARACTER (KB);
  if (LETTER >= 'A') and (LETTER <= 'B') then do;
    do case LETTER;
    call API_TEST;
    call MIDS_TEST;
    end; /* case */
  end; /* (LETTER >= 'A') and (LETTER <= 'B') */
  if LETTER <> ESC then
    call WRITE_OUT (@CONSOLE_MENU);
  end;

end CONSOLE_DRIVER;

end API_APPLICATIONS;
2.3.1.2 ASM86 files

2.3.1.2.1 RM_SEGS.ASM

$ debug
$ xref

; /******************************************************************************
; * MODULE NAME : REAL_MODE_SEGMENTS *
; */
 /******************************************************************************/

; Source Filename : RM_SEGS.ASM
; Source Compiler : ASM86
; Operating System : DOS 3.10
; Description : Defines segments for the real mode code to enable absolute locating.
; Public procedures: None
; EPD files : None
; Include files : None

$ eject

; /******************************************************************************/
; HISTORY Version 1.0 :
; Designed by : P.A. OLANDER Date : November 1989
; Description : Original
; /******************************************************************************/
TEST_RESULTS struc
  CPU_BOARD  db ?
  CPU        db ?
  ROM        db ?
  MASTER_PIC db ?
  SLAVE_PIC  db ?
  TMOUT_CIRCUIT db ?
  PIT        db ?
  COPROCESSOR db ?
  MPCSC      db ?
  RAM        db ?
  ACCESS_TO_1611 db ?
  ERAM       db ?
  EROM       db ?
  SDB_BOARD  db ?
  SDB_SELFTEST db ?
  SDB_RAM    db ?
  LOCAL_BUS  db ?
  API_BOARD  db ?
  API_RAM    db ?
  API_EMAC   db ?
  EMAC_SWITCH db 8 dup (?)
  STUCK_MODULE db ?
  STUCK_KEY  db ?
  DRAM_MBUS  db ?
  DRAM_LBUS  db ?
  SLAVE      db 4 dup (?)
TEST_RESULTS ends

TEST_RESULTS_SEG segment public
TEST_RESULT TEST_RESULTS <-
public     TEST_RESULT
TEST_RESULTS_SEG ends
TMAP_RESULTS struct
TMAP_BOARD dw ?
TMAP_CPU dw ?
TMAP_ROM dw ?
TMAP_MASTER_PIC dw ?
TMAP_SLAVE_PIC dw ?
TMAP_TMOUT_CIRCUIT dw ?
TMAP_PIT dw ?
TMAP_COPROCESSOR dw ?
TMAP_MPSC dw ?
TMAP_RAM dw ?
TMAP_ACCESS_TO_1611 dw ?
TMAP_ERAM dw ?
TMAP_EROM dw ?
TMAP_SDB_BOARD dw ?
TMAP_SDB_SELFTEST dw ?
TMAP_SDB_RAM dw ?
TMAP_LOCAL_BUS dw ?
TMAP_API_BOARD dw ?
TMAP_API_RAM dw ?
TMAP_DRAM_MBUSBUS dw ?
TMAP_DRAM_LBUS dw ?
TMAP_SLABE dw 4 dup (?)
TMAP_RESULTS ends

TMAP_GLOBAL_SEG segment
TMAP_FLAG dw ?
TMAP_SCREEN dw ?
TMAP_RESULT TMAP_RESULTS <>
public TMAP_FLAG
public TMAP_SCREEN
public TMAP_RESULT
TMAP_GLOBAL_SEG ends

SLAVE_RESULTS struct
SLAVE_BOARD dw 4 dup (?)
SLAVE_RESULTS ends

GRAPHICS_BUFFERS struct
BUFFER_FLAG dw ?
DATA_TYPE dw ?
GPR_BUSY dw ?
GRAPHICS_CONTENTS db 2045 dup (?)
GRAPHICS_BUFFERS ends

SCMB_BUFFERS struct
SCMB_CONTENTS db 2048 dup (?)
SCMB_BUFFERS ends
APISEG_1 segment
API_RAM_BUFFER db 7fh dup (?)
public API_RAM_BUFFER
APISEG_1 ends

APISEG_2 segment
SKM1_CONTROL db 80h dup (?) ; BASE + 1000h - 107FH
SKM2_CONTROL db 80h dup (?) ; BASE + 1080h - 10FFH
SKM3_CONTROL db 80h dup (?) ; BASE + 1100h - 117FH
QKBM_CONTROL db 80h dup (?) ; BASE + 1180h - 11FFH
FILLER_1 db 80h dup (?) ; BASE + 1200h - 127FH
EMAC_CONTROL db 80h dup (?) ; BASE + 1280h - 12FFH
SPARE_CONTROL db 80h dup (?) ; BASE + 1300h - 137FH
SPARE_STATUS db 80h dup (?) ; BASE + 1380h - 13FFH
KEY_STATUS db 80h dup (?) ; BASE + 1400h - 147FH
QKBM_STATUS db 80h dup (?) ; BASE + 1480h - 14FFH
RBM_STATUS db 80h dup (?) ; BASE + 1500h - 157FH
EMAC_STATUS db 80h dup (?) ; BASE + 1580h - 15FFH
FILLER_2 db 100h dup (?) ; BASE + 1600h - 16FFH
FILLER_3 db 0fh dup (?) ; BASE + 1700h - 17FFH
API_WR_DATA dw ? ; BASE + 17FCh - 17FFH
AP_WR_DATA dw ? ; BASE + 17FEh - 17FFH
public SKM1_CONTROL
public SKM2_CONTROL
public SKM3_CONTROL
public QKBM_CONTROL
public EMAC_CONTROL
public SPARE_CONTROL
public SPARE_STATUS
public KEY_STATUS
public QKBM_STATUS
public RBM_STATUS
public EMAC_STATUS
public API_WR_DATA
public AP_WR_DATA
APISEG_2 ends

end
2.3.1.2.2 RM_COPY.ASM

$ debug
$ xref

; /************************************************************************** *
; * MODULE NAME : COPY_PROG *
; * /************************************************************************** *
; * Source Filename : RM_COPY.ASM *
; * Source Compiler : ASM86 *
; * Operating System : DOS 3.10 *
; * Description : The program first copies the descriptor tables residing on EPROM into RAM and executes an absolute jump to the code that initialises the descriptor table registers and sets the status to 80286 protected virtual address mode. *
; * Public procedures: EXCHANGE_FLAG *
; * EPD files : None *
; * Include files : None *

$ eject

; /************************************************************************** *
; * HISTORY Version 1.0 : *
; * Designed by : P.A. OLANDER Date : February 1990 *
; * Description : Original *

; /**************************************************************************/
name COPY_PROG

ROM_BASE_1 equ 0dd00h ; ROM base address of descriptor tables for first processor card
ROM_BASE_2 equ 0db00h ; ROM base address of descriptor tables for second processor card
RAM_BASE equ 0h ; RAM base address of descriptor tables
LIMIT equ 2000h ; Length of descriptor tables
MAP equ 5h ; Used to distinguish between CPU cards

SYSTEM_DESC segment public 'SYSTEM_DESC'
extrn NO_OF_PROCS : byte
extrn PMODE_VER : byte
SYSTEM_DESC ends

PROC_ID_SEG segment
PROC_ID dw ?
NO_OF_RUNS dw ?
RUN_ID dw ?
public PROC_ID
public NO_OF_RUNS
public RUN_ID
PROC_ID_SEG ends

GLOBAL_RAM segment
EXCHANGED_FLAG dw ?
SYNCHRONISATION_FLAG dw ?
STATUS_FLAG dw ?
public EXCHANGED_FLAG
public SYNCHRONISATION_FLAG
public STATUS_FLAG
GLOBAL_RAM ends

ENTP_DATA segment at 850h ; Storage for stack pointer when entering Protected Mode
TEMP_VAR db ?
ENTP_DATA ends
COPY_CODE segment public 'COPY_DESCRIPTORS'
assume CS:COPY_CODE, DS:SYSTEM_DESC

ENTP_CODE dd 0AFF00000h ; Absolute location of code to enter
                      ; Protected Mode

publ ic COPY_TABLES
COPY_TABLES: cli       ; Disable all interrupts
             mov AX,SYSTEM_DESC
             mov DS,AX

             mov AL,PMODE_VER ; Code on EPROM ?
             cmp AL,1        ; If so then copy descriptor tables
             jne PROT_MODE  ;         else do not copy tables

             mov AL,NO_OF_PROCS ; How many processors in this system ?
             cmp AL,1        ; If only one, then
             je FIRST_TABLE ;          just copy first table (dd000)

             mov AX,PROC_ID_SEG ; Has the card identified itself as
             mov DS,AX
             cmp DS:RUN_ID,MAP ; the primary or secondary processor
             je FIRST_TABLE ;         If primary, then this card is Master

             mov AX,ROM_BASE_2 ; else this card is slave so
             mov DS,AX
             jmp COPY ; copy second table (db000) to RAM

FIRST_TABLE:  mov AX,ROM_BASE_1 ; Load ROM address into DS register
             mov DS,AX
COPY:        mov AX,RAM_BASE ; Load RAM address into ES register
             mov ES,AX
             xor DI,DI ; Clear destination index register
             xor SI,SI ; Clear source index register
             mov CX,LIMIT
cld             ; Get ready to increment pointers

             rep movsw

PROC_MODE:   mov AX,ENTP_DATA ; Set up data segment for
             mov DS,AX ; Protected Mode
             jmp dword ptr ENTP_CODE ; Absolute jump to predefined address
                                 ; (CS = AFF0 and IP = 0)
EXCHANGE_FLAG proc far

public EXCHANGE_FLAG

FIRST_HERE equ 5a5ah

push DS
push ES
cli
mov AX,GLOBAL_RAM ; Reset DS to point to global RAM
mov DS,AX

; mov AX,FIRST_HERE ; Get ready to set global flag
; xchg AX,DS:EXCHANGED_FLAG ; Set global flag

; Since the API hardware does not support the bus lock facility,
; the read-modify-write sequence has to be implemented.

mov BX,FIRST_HERE
mov AX,DS:EXCHANGED_FLAG ; Read global location
mov DS:EXCHANGED_FLAG,BX ; Write to global location
sti
pop ES
pop DS
ret

EXCHANGE_FLAG endp

COPY_CODE ends

end
2.3.1.3 Entities publicly declared files

2.3.1.3.1 RM_APP.EPD

SYSTEM_APPLICATIONS: procedure external;
   end SYSTEM_APPLICATIONS;

2.3.1.3.2 RM_BIT.EPD

declare EMAC_SWITCH (8) byte external;

RM_BLOCK_TEST: procedure (BLOCK_BASE_PTR, NO_OF_WORDS) byte external;
   declare NO_OF_WORDS word;
   declare BLOCK_BASE_PTR pointer;
   end RM_BLOCK_TEST;

TEST_API_LOCATIONS: procedure external;
   end TEST_API_LOCATIONS;

API_RESET_TEST: procedure external;
   end API_RESET_TEST;

API_SELF_TEST: procedure external;
   end API_SELF_TEST;

EMAC_TEST: procedure external;
   end EMAC_TEST;
declare KEYBOARD_INTERRUPT byte external;
declare KEYBOARD_INPUT byte external;

declare API_INTERRUPT byte external;
declare STUCK_KEY_DETECTED byte external;
declare QKBM_KEY_PRESSED byte external;
declare RBM_MOVED byte external;
declare EMAC_INTERRUPT byte external;
declare SPARE_INTERRUPT byte external;
declare API_DATA (10) word external;
declare API_DATA_START_INDEX byte external;
declare API_DATA_END_INDEX byte external;

declare DPR_INTERRUPT byte external;
declare DPR_RX byte external;
declare DPR_RX_HI byte external;

/*****************************************************************************/
* * Master PIC interrupt handlers *
* */

CHB_RX: procedure interrupt 82 external;

/*****************************************************************************/
* * This interrupt handler is used for serial communications during the API *
* on-line demonstration, rather than the standardised handler. It simply *
* reads a character received from the test terminal and sets an indicator *
* that a character has been received. *
* */

end CHB_RX;
C-113

Slave PIC interrupt handlers

API: procedure interrupt 128 external;

Handles interrupts from the API card during POST and off-line diagnostics.

end API;

DPR_INTERRUPT_READ : procedure interrupt 128 external;

Handles interrupts from the API card during the on-line API functionality demonstration. It reads the location written to by the API and sets a flag.

end DPR_INTERRUPT_READ;

2.3.1.3.4 RM_MSCC.EPD

MASS_STORAGE_TESTS : procedure external;
end MASS_STORAGE_TESTS;

2.3.1.3.5 RM_API.EPD

CONSOLE_DRIVER: procedure external;
end CONSOLE_DRIVER;
declare TEST_RESULT structure

(CPU_BOARD byte,
CPU byte,
ROM byte,
MASTER_PIC byte,
SLAVE_PIC byte,
TMOUT_CIRCUIT byte,
PIT byte,
COPROCESSOR byte,
MPSC byte,
RAM byte,
ACCESS_TO_1611 byte,
ERAM byte,
EROM byte,
SDB_BOARD byte,
SDB_SELFTEST byte,
SDB_RAM byte,
LOCAL_BUS byte,
API_BOARD byte,
API_RAM byte,
API_EMAC byte,
EMAC_SWITCH byte,
STUCK_MODULE byte,
STUCK_KEY byte,
DRAM_MBUS byte,
DRAM_LBUS byte,
SLAVE byte ) external;
declare TMAP_RESULT structure

    (TMAP_BOARD byte,  
      TMAP_CPU byte,  
      TMAP_ROM byte,  
      TMAP_MASTER_PIC byte,  
      TMAP_SLAVE_PIC byte,  
      TMAP_TMOUT_CIRCUIT byte,  
      TMAP_PIT byte,  
      TMAP_COProcessor byte,  
      TMAP_MPSC byte,  
      TMAP_RAM byte,  
      TMAP_ACCESS_TO_1611 byte,  
      TMAP_ERAM byte,  
      TMAP_EROM byte,  
      TMAP_SDB_BOARD byte,  
      TMAP_SDB_SELFTEST byte,  
      TMAP_SDB_RAM byte,  
      TMAP_LOCAL_BUS byte,  
      TMAP_API_BOARD byte,  
      TMAP_API_RAM byte,  
      TMAP_DRAM_MBUS byte,  
      TMAP_DRAM_LBUS byte,  
      TMAP_SLAVE (4 byte) external;

declare TMAP_FLAG byte external;
declare TMAP_SCREEN byte external;

declare SLAVE_RESULT structure

    (SLAVE_BOARD (4 byte) external;

declare GRAPHICS_BUFFER (3) structure

    (BUFFER_FLAG byte,  
      DATA_TYPE byte,  
      GPR_BUSY byte,  
      GRAPHICS_CONTENTS (2045) byte ) external;

declare SCMB_BUFFER (2) structure

    (SCMB_CONTENTS (2048) byte ) external;

declare API_RAM_BUFFER (7fch) byte external;
declare SKM1_CONTROL (80h) byte external; /* BASE + 1000h - 107FH */
declare SKM2_CONTROL (80h) byte external; /* BASE + 1080h - 10FFH */
declare SKM3_CONTROL (80h) byte external; /* BASE + 1100h - 117FH */
declare QKBM_CONTROL (80h) byte external; /* BASE + 1180h - 11FFH */
declare FILLER_1 (80h) byte external; /* BASE + 1200h - 127FH */
declare EMAC_CONTROL (80h) byte external; /* BASE + 1280h - 12FFH */
declare SPARE_CONTROL (80h) byte external; /* BASE + 1300h - 137FH */
declare SPARE_STATUS (80h) byte external; /* BASE + 1380h - 13FFH */
declare KEY_STATUS (80h) byte external; /* BASE + 1400h - 147FH */
declare QKBM_STATUS (80h) byte external; /* BASE + 1480h - 14FFH */
declare RBM_STATUS (80h) byte external; /* BASE + 1500h - 157FH */
declare EMAC_STATUS (80h) byte external; /* BASE + 1580h - 15FFH */
declare FILLER_2 (100h) byte external; /* BASE + 1600h - 167FH */
declare FILLER_3 (0Fch) byte external; /* BASE + 1700h - 177FH */
declare API_WR_DATA word external; /* BASE + 17FCh - 17FFh */
declare AP_WR_DATA word external; /* BASE + 17FFh - 17FFh */

2.3.1.3.7 RM_COPY.EPD

declare PROC_ID byte external;
declare NO_OF_RUNS byte external;
declare RUN_ID byte external;
declare EXCHANGED_FLAG byte external;
declare SYNCHRONISATION_FLAG byte external;
declare STATUS_FLAG byte external;
declare COPY_TABLES label external;

EXCHANGE_FLAG: procedure byte external;
    end EXCHANGE_FLAG;
2.3.1.4 Included files

2.3.1.4.1 STAT.INC

/***********************************************~~*******************
* Subsystem-specific status latch literals *
* The values listed below are those which are output to the 8-bit *
* diagnostic latch of either processor card in the event of a *
* failure occurring during the Power On Self Tests or off-line *
* diagnostics. *
***********************************************~~******************/

/* API tests */

declare API_DPR_TEST_NO literally '40h' ;
declare API_RESET_NO literally '41h' ;
declare API_CHECKSUM_ACK literally '42h' ;
declare API_CHECKSUM_NO literally '43h' ;
declare API_POWER_UP_NO literally '44h' ;
declare MIDS_RESET_NO literally '45h' ;
declare API_SELF_TEST_NO literally '46h' ;
declare EMAC_TEST_NO literally '47h' ;
declare NO_GRAPHICS_CARDS literally '60h' ;
declare MBUS_DRAM_FAILURE literally '70h' ;

/* General status latch outputs */
declare PROC_TOO_LATE literally '0f2h' ;
/* Console demonstration literal declarations */

declare FOREVER literally 'while 1';
declare TRUE literally '0ffh';
declare SET literally '0ffh';
declare RESET literally '0';
declare FALSE literally '0';
declare KB literally '1';
declare VDU literally '1';
declare PASSED literally '2';
declare FAILED literally '1';
declare ABORTED literally '0ffh';
declare DIS_CLR literally '1bh,3ch,1bh,5bh,31h,3bh,31h,48h,1bh,5bh,32h,4ah';
   /* esc < esc [ 1 ; 1 H esc I 2 J*/
declare DIS_NME literally '1bh,5bh,31h,3bh,31h,48h';
   /* esc [ 1 ; 1 H */
declare CLR_SCREEN literally '1bh,5bh,4ah';
   /* esc [ j */
declare CUR_UP literally '1bh,4dh,1bh,5bh,31h,4ah';
   /* esc M esc [ 1 j */
declare RES_AREA literally '1bh,5bh,31h,32h,3bh,31h,48h,1bh,5bh,4bh';
   /* esc [ 1 2 ; 1 H esc I K*/
declare POINT_AREA literally '1bh,5bh,31h,32h,3bh,31h,48h';
   /* esc [ 1 2 ; 1 H */
declare EMAC_AREA literally '1bh,5bh,31h,34h,3bh,31h,48h,1bh,5bh,4bh';
   /* esc [ 1 4 ; 1 H esc I K*/
declare SKM1_AREA literally '1bh,5bh,31h,35h,3bh,31h,48h,1bh,5bh,4bh';
   /* esc [ 1 5 ; 1 H esc I K*/
declare SKM2_AREA literally '1bh,5bh,31h,36h,3bh,31h,48h,1bh,5bh,4bh';
   /* esc [ 1 6 ; 1 H esc I K*/
declare SKM3_AREA literally '1bh,5bh,31h,37h,3bh,31h,48h,1bh,5bh,4bh';
   /* esc [ 1 7 ; 1 H esc I K*/
declare QEM AREA literally '1bh,5bh,31h,38h,3bh,31h,48h,1bh,5bh,4bh';
   /* esc [ 1 8 ; 1 H esc I K*/
declare UNKN_AREA literally '1bh,5bh,31h,39h,3bh,31h,48h,1bh,5bh,4bh';
   /* esc [ 1 9 ; 1 H esc I K*/
declare RBM_AREA literally '1bh,5bh,32h,30h,3bh,31h,48h,1bh,5bh,4bh';
   /* esc [ 2 0 ; 1 H esc I K*/
declare STAT_AREA literally '1bh,5bh,32h,32h,3bh,31h,48h,1bh,5bh,4bh';
   /* esc [ 2 2 ; 1 H esc I K*/
declare STAT1_AREA literally '1bh,5bh,32h,33h,3bh,31h,48h,1bh,5bh,4bh';
   /* esc [ 2 3 ; 1 H esc I K*/
declare STAT2_AREA literally '1bh,5bh,32h,34h,3bh,31h,48h,1bh,5bh,4bh';
   /* esc [ 2 4 ; 1 H esc I K*/
declare TEST_AREA literally '1bh,5bh,32h,33h,3bh,31h,48h,1bh,5bh,4bh';
   /* esc [ 2 3 ; 1 H esc I K*/
declare MASK DEVICE literally '20h';
declare UNMASK DEVICE literally '30h';
declare STUCK_ENABLE literally '60h';
declare STUCK_DISABLE literally '70h';
declare API_READ_ACKNOWLEDGE literally '20h';
declare AP_READ_ACKNOWLEDGE literally '90h';
declare VALID_API_RESPONSE literally '90h';
declare EMAC_BASE literally '0';
declare SKM1_BASE literally '1';
declare SKM2_BASE literally '2';
declare SKM3_BASE literally '3';
declare QKBMBASE literally '4';
declare SPARE_BASE literally '5';
declare CPU_ADDR literally '11000h';

/** 286_CPU I/O PORT ADDRESSES */
declare WR_REG1_8259 literally '0c0h'; /* Interrupt controller WRITE reg */
declare WR_REG2_8259 literally '0c2h'; /* Interrupt controller WRITE reg */
declare WR_REG1_8259S literally '0c4h'; /* Interrupt controller WRITE reg */
declare WR_REG2_8259S literally '0c6h'; /* Interrupt controller WRITE reg */
declare NO_POLL literally '00001000b'; /* OCW_3 no poll command */
declare DATA_PORT_8274A literally '0d0h'; /* Serial controller Data_port */
declare CNTRL_REG_8274A literally '0dch'; /* Serial controller WRITE reg */
declare DATA_PORT_8274B literally '0dah'; /* Serial controller Data_port */
declare CNTRL_REG_8274B literally '0deh'; /* Serial controller WRITE reg */
declare TIMER_0_8254 literally '0d0h'; /* Counter/timer reg Counter_0 */
declare TIMER_1_8254 literally '0d2h'; /* Counter/timer reg Counter_1 */
declare TIMER_2_8254 literally '0d4h'; /* Counter/timer reg Counter_2 */
declare CNTRL_REG_8254 literally '0d6h'; /* Counter/timer control register */
2.3.1.4.3 MSCC.INC

/ * MSCC Commands */

declare READ_FILE literally 'OFFh';
declare WRITE_FILE literally 'OF Eh';
declare READ_DIRECTORY literally 'OF Dh';
declare RENAME_FILE literally 'OF Ch';
declare COPY_FILE literally 'OF Bh';
declare TEST_BOARD literally 'OF Ah';
declare PRINT_FILE literally 'OF 9h';
declare PRINT_BLOCK literally 'OF 8h';
declare INITIALISE_SERIAL_PORT literally 'OF 7h';
declare FORMAT_DISK literally 'OF 6h';
declare DELETE_FILE literally 'OF 5h';
declare APPEND_FILE literally 'OF 4h';
declare CONTINUE_PRINT literally 'OF 3h';
declare LIST_PARTITION literally 'OF 2h';
declare MAKE_PARTITION literally 'OF 1h';
declare REMOVE_PARTITION literally 'OF 0h';
declare HDISK_FORMAT literally '0E Fh';
declare ABORT_PRINT literally '0EEh';

/ * MSCC Drive types */

declare SERIAL_PORT_A literally '00h';
declare SERIAL_PORT_B literally '01h';
declare PRINTER_PORT literally '02h';
declare FLOPPY_DISK_0 literally '03h';
declare FLOPPY_DISK_1 literally '04h';
declare FLOPPY_DISK_2 literally '05h';
declare FLOPPY_DISK_3 literally '06h';
declare FIXED_DISK_0 literally '07h';
declare FIXED_DISK_1 literally '08h';
declare FIXED_DISK_2 literally '09h';
declare FIXED_DISK_3 literally '0Ah';
/* MSCC Error number / Status values */

/* STATUS VALUES */

declare OKAY_STATUS literally '0';
declare INVALID_TRACK_STATUS literally '1';
declare INVALID_SECTOR_STATUS literally '2';
declare TRACK_NOT_FOUND_STATUS literally '3';
declare DNA_TYPE_INCORRECT_STATUS literally '4';
declare FILE_NOT_FOUND_STATUS literally '5';
declare FILE_TRUNCATED_STATUS literally '6';
declare NO_SPACE_ON_DISK_STATUS literally '7';
declare FILE_EXISTS_STATUS literally '8';
declare NO_ROOT_DIRECTORY_SPACE_STATUS literally '9';
declare BUSY_STATUS literally '10';
declare INVALID_BAUD_RATE_STATUS literally '11';
declare INVALID_NO_OF_DATA_BITS_STATUS literally '12';
declare INVALID_PORT_NO_STATUS literally '13';
declare MAX_FILE_LENGTH_EXCEEDED_STATUS literally '14';
declare PAPER_OUT_STATUS literally '15';
declare OFF_LINE_STATUS literally '16';
declare INVALID_SERIAL_PORT_STATUS literally '17';
declare INVALID_HANDSHAKING_STATUS literally '18';
declare INVALID_SERIAL_interrupt_STATUS literally '19';
declare NOT_IMPLEMENTED_STATUS literally '20';
declare INVALID_COMMAND_STATUS literally '21';
declare INVALID_SCSI_COMMAND_STATUS literally '22';
declare NO_BUFFER_SPACE_STATUS literally '23';
declare ERROR_IN_FILENAME_STATUS literally '24';
declare DEVICE_NOT_CONNECTED_STATUS literally '25';
declare INVALID_ADDRESS_STATUS literally '26';
declare DATA_WAITING_STATUS literally '27';
declare INVALID_FORMAT_STATUS literally '28';
declare INVALID_PARTITIONSTATUS literally '29';
declare PARTITION_EXISTS_STATUS literally '30';
declare NO_PARTITION_TABLE_SPACE_STATUS literally '31';
declare INVALID_DEVICE_STATUS literally '32';
declare FLOPPY_NOT_READY_STATUS literally '33';

/* Test parameters */

declare FULL_TEST literally '0';
declare CPU_TEST literally '1';
declare RAM_TEST literally '2';
/* Baud rate values */
declare B_50 literally '0';
declare B_75 literally '1';
declare B_110 literally '2';
declare B_150 literally '3';
declare B_300 literally '4';
declare B_600 literally '5';
declare B_1200 literally '6';
declare B_1800 literally '7';
declare B_2400 literally '8';
declare B_3600 literally '9';
declare B_4800 literally '10';
declare B_7200 literally '11';
declare B_9600 literally '12';
declare B_14400 literally '13';
declare B_19200 literally '14';
/* Disk type values for FORMATTING */
declare FLOPPY_360K literally '0';
declare STIFFY_720K literally '1';
declare HARD_DISK literally '2';
/* Serial Interrupt Mode */
declare NO_INTERRUPTS literally '0000000000b';
declare TX_BUFFER_EMPTY_INTERRUPT literally '0000000005b';
declare BLOCK_RECEIVED_INTERRUPT literally '0000000010b';
declare CHARACTER_RECEIVED_INTERRUPT literally '0000000011b';
/* Handshaking Mode */
declare NO_HANDSHAKING literally '0000000000b';
declare RTS_CTS_HANDSHAKING literally '0000000010b';
declare XON_XOFF_HANDSHAKING literally '0000000011b';
/* Serial Port Mode */
declare MONITOR_TERMINAL literally '0000000000b';
declare DATA_TERMINAL literally '0000000011b';
/* Null pointer for serial buffers */
declare NUL literally '0000000000b';
/* Hard disk type */
declare DOS_DISK literally '0';
declare OPTICAL_DISK literally '1';
/* Addresses on MULTIBUS for MSCC */

/* Port addresses */

declare MSCC_GPR literally '600h';
declare MSCC_STATUS_REGISTER literally '601h';
declare MSCC_RESET_SLAVE_INT literally '602h';
declare MSCC_RESET_BOARD literally '603h';

/* Memory Addresses */
declare MSCC_PARAMETER_BLOCK_ADDRESS dword at ( 30000h );
declare MSCC_RESPONSE_BLOCK_ADDRESS dword at ( 30004h );

2.3.1.4.4 CONSMPSC.INC

declare CHANNEL_RESET literally '00011000b';

/* Channel A */
declare REG_4_VAL literally '01000100b';
/* reg 4, x16 clock mode, 1 stop bit, parity disabled */
declare REG_1A_VAL literally '00000000b';
/* Rx interrupt disabled */
declare REG_2A_VAL literally '00110000b';
/* System configuration */
declare REG_5A_VAL literally '01100000b';
/* Tx 8 bits/ch, tx disabled */
declare REG_3A_VAL literally '11000000b';
/* Rx disable */
declare REG_1B_VAL literally '00011100b';
/* Interrupt on all characters */

declare REG_2B_VAL literally '01010000b';
/* Interrupt vector = 80 */

declare REG_5B_VAL literally '11100010b';
/* Tx 8 bits/ch, tx disabled */

declare REG_3B_VAL literally '1100001b';
/* Rx enable */

2.3.1.4.5 DPRMEM.INC

declare SKM1_CONT structure (SKM(2) byte) at (CPU_ADDR);
declare SKM2_CONT structure (SKM(2) byte) at (CPU_ADDR + 80h);
declare SKM3_CONT structure (SKM(2) byte) at (CPU_ADDR + 100h);
declare QKBM_CONT structure (QKBM(2) byte) at (CPU_ADDR + 180h);
declare GDU_CONT structure (GDU(7fh) byte) at (CPU_ADDR + 200h);
declare EMAC_CONT structure (EMAC(2) byte) at (CPU_ADDR + 280h);
declare SPARE_CONT structure (SPARE(2) byte) at (CPU_ADDR + 300h);

declare API_INTERRUPT_LOCATION byte at (CPU_ADDR + 7FEh);
/* The API is interrupted on a CPU write to this location */
2.3.2 Protected mode code

2.3.2.1 PL/M 286 files

2.3.2.1.1 PM_ENTP.PLM

$ include (PLMPAR.INC)

/***************************************************************************
* *
* MODULE NAME : ENTER_PROTECTED_MODE *
* *
***************************************************************************
*
* Source Filename : PM_ENTP.PLM *
* *
* Source Compiler : PLM286 *
* *
* Operating System : DOS 3.10 *
* *
* Description : Routine to cause the transition from *
* real mode to protected mode. *
* *
* Public procedures: None *
* *
* EPD files : None *
* *
* Include files : PLMPAR.INC *
* *
***************************************************************************

***************************************************************************
*
* HISTORY Version 1.0 : *
* *
* Designed by : P.A. OLANDER Date : February 1989 *
* Description : Original *
* *
***************************************************************************
ENTER_PROTECTED_MODE: do;

declare BEGIN label public;
   /* Included to generate code to set up segment selectors */

declare SYSTEM_EXECUTIVE label external;
   /* Jump to this task once protected mode has been entered */

declare PMODE_LATCH literally '0cah' ;
   /* CPU card Protected Mode latch address */

declare PMODE_COPROC literally '03h' ;

declare TRUE literally '0ffffh';

declare TASK_REG_VAL literally '020h' ;
   /* Initial task register: this is the selector to be put into the task register pointing to GDT(4), i.e. the currently executing task */

declare LDTR_VAL literally '18h' ;
   /* Initial LDTR: this is the selector value to be put into the local descriptor table register pointing to GDT(3), i.e. the position in the GDT where the LDT descriptor is stored */

declare GDTR_VAL structure
   (LIMIT word, BASE dword, ACCESS byte ) ;

declare IDTR_VAL structure
   (LIMIT word, BASE dword, ACCESS byte) ;
disable;

GDTR_VAL.LIMIT = 31fh;
GDTR_VAL.BASE = 800h;
GDTR_VAL.ACCESS = 0;
IDTR_VAL.LIMIT = 7ffh;
IDTR_VAL.BASE = 0h;
IDTR_VAL.ACCESS = 0;

MACHINE$STATUS = (MACHINE$STATUS or PMODE_COPROC);
/* Set protected mode and enable numeric coprocessing */

outword(PMODE_LATCH) = TRUE ;
/* Set the latch on the CPU board */

call RESTORESGLOBALSTABLE (GDTR_VAL);
/* Initialise Global Descriptor Table Register */

call RESTORESINTERRUPTSTABLE (IDTR_VAL);
/* Initialise Interrupt Descriptor Table Register */

LOCAL$TABLE = SELECTOR (LDTR_VAL);
/* Initialise Local Descriptor Table Register */

TASKSREGISTER = SELECTOR (TASK_REG_VAL);
/* Initialise Task Register */

goto SYSTEM_EXECUTIVE;
/* Performs a task switch to the executive routine */

end ENTER_PROTECTED_MODE;
2.3.2.1.2 PM_EXEC.PLM

$ include (PLMPAR.INC)

/*****************************************************************************
  *
  * MODULE NAME : PROTECTED_MODE_EXECUTIVE
  *
  *****************************************************************************/

  *
  * Source Filename : PM_EXEC.PLM
  *
  * Source Compiler : PLM286
  *
  * Operating System : DOS 3.10
  *
  * Description : Executive driving routine for
  * the real mode applications code.
  *
  * Public procedures: None
  *
  * EPD files : STDIO.EPD
  * PM_INIT.EPD
  * PM_APP.EPD
  *
  * Include files : PLMPAR.INC
  * LITS.INC
  * PICLITS.INC
  *
  *****************************************************************************/

*******************************************************************************/

  *
  * HISTORY Version 1.0 :
  *
  * Designed by : P.A. OLANDER  Date : September 1989
  * Description : Original
  *
  *****************************************************************************/
PROTECTED_MODE_EXECUTIVE: do;

declare IO_BASE_ADDRESS literally '0dh';
declare IO_INTERRUPT_NO literally '80';
declare IO_BAUD_RATE literally '9600';

$ include (\..\..\..\STD\STDDIO.EPD)
$ include (LITS.INC)
$ include (\..\..\..\STD\PICLITS.INC)
$ include (PM_INIT.EPD)
$ include (PM_APP.EPD)

declare SYSTEM_EXECUTIVE label public;
declare DO_POST label public;

declare INITIALISATION_FINISHED byte external;
declare FOREVER byte;

INITIALISE_PORT_DATA: procedure;

call INITIALISE_IO_ADDRESSES (PORT_A,
    IO_BASE_ADDRESS,
    IO_INTERRUPT_NO);

call INITIALISE_PORT( PORT_A,
    IO_BAUD_RATE,
    EIGHT_DATA_BITS,
    TWO_STOP_BITS,
    NO_PARITY,
    TERMINAL);

call INITIALISE_PORT( PORT_B,
    IO_BAUD_RATE,
    EIGHT_DATA_BITS,
    TWO_STOP_BITS,
    NO_PARITY,
    TERMINAL);

end INITIALISE_PORT_DATA;
SYSTEM_EXECUTIVE:

call CLEAR$TASK$SWITCHED$FLAG;
INITIALISATION_FINISHED = FALSE;

output (PIC_MASTER_8259A_ADR2) = MASK_ALL_INTS;
output (PIC_SLAVE_8259A_ADR2) = MASK_ALL_INTS;

call INITIALISE_PORT_DATA; /* Set up port data for PVAM addressing */

DO_POST:

call PM_INITIALISATION;

FOREVER = TRUE;
do while FOREVER = TRUE;

call MAIN;

end;

end PROTECTED_MODE_EXECUTIVE;
$ include (PLMPAR.INC)

/***************************************************************************/
* Module Name : PROTECTED_MODE_INITIALISATION
* *
***************************************************************************/

* Source Filename : PM_INIT.PLM
* *
* Source Compiler : PLM286
* *
* Operating System : DOS 3.10
* *
* Description : Protected mode initialisation and POST routines
* *
***************************************************************************/

* Public procedures: DISPLAY
  POST_DRAM_TEST
  PM_INITIALISATION
* *
* EPD files : STDIO.EPD
  PM_SEGS.EPD
* *
* Include files : PLMPAR.INC
  PICLITS.INC
  STATLITS.INC
  STAT.INC
* *
***************************************************************************/

*******************************************************************************/
* HISTORY Version 1.0 :*
* *
* Designed by : P.A. OLANDER Date : October 1990
* *
*******************************************************************************/
PROTECTED_MODE_INITIALIZATION: do;

$ include (.\STD\STDIO.EPD)
$ eject
$ include (.\STD\PICLITS.INC)
$ eject
$ include (.\STD\STATLITS.INC)
$ eject
$ include (.\REALMODE\STAT.INC)
$ eject
$ include (PM_SEG.SEPD)
$ eject

declare ZERO_ONES literally '55h';
declare ONE_ZEROS literally '0ah';
declare DRAM_BUFFER_MAX literally '0fffh';
declare DRAM_BUFFER_MIN literally '0fh';
declare UTESTED literally '0';
declare PASSED literally '1';
declare PRESENT literally '1';
declare RAM_FAILURE literally '2';
declare CPU_FAILURE literally '3';
declare FAILED literally '2';
declare NOT_PRESENT literally '3';
declare ABSENT literally '8';
declare REPORT_TO_EPROM_RAM literally '3';
declare REPORT_TO_DRAM literally '9';
declare FALSE literally '0';
declare TRUE literally '1';
declare BUSY literally '1';
declare RESET literally '0';
declare TMAP literally '0ah';
declare TMAP_PRESENT literally '0ah';
declare INTERROGATE_TMAP literally '51h';
declare GIM_SLOT_1 literally '17';
declare INT_PENDING_MASK literally '00000010b';
declare GIM_ERROR_MASK literally '00000010b';
declare SCMB_ADDRESS literally '500h';
declare SLAVE_ADDRESS_1 literally '400h';
declare SLAVE_ADDRESS_2 literally '300h';
declare SLAVE_ADDRESS_3 literally '200h';
declare TEXT literally '1';
declare MBUS literally '05h';
declare LBUS literally '0ah';
declare GET literally '0';
declare PUT literally '1';
declare MAX_LOOP_COUNTER literally '10';
declare VDU literally '1';
declare KB literally '1';
declare ERASE_SCREEN (*) byte data (ESC, '072', ESC, '01';OH', EOM);
   /* Erases the screen and homes the cursor */

declare TEST byte external;

declare RECEIVE_CONTROL byte external;
declare RECEIVE_DATA byte external;
declare TRANSMIT_CONTROL byte external;
declare TRANSMIT_DATA (100h) byte external;
declare DUAL_PORT_RAM byte external;
declare TMOUT_INTERRUPT byte external;

declare PROCESSOR_ID byte external;

declare DRAM_BUFFER_PTR pointer;
declare DRAM_BUFFER based DRAM_BUFFER_PTR (DRAM_BUFFER_MAX) byte;

declare INITIALISATION_FINISHED byte public;
declare BOARD_NO byte public;
declare DIM_AVAILABLE byte public;
declare SCREENS_AVAILABLE byte public;
declare BIT_INVOKED byte public;
declare TMAP_PRESENCE byte public;
declare MIDS_START_INDEX byte public;
declare MIDS_END_INDEX byte public;

declare DRAM_TEST_RESULT byte;
declare GPR_ADDRESS word;

GPR_BUSY: procedure (SCREEN_NO);
   /* Procedure waits for specified */
   declare SCREEN_NO byte; /* graphics card to finish task */
   /* then sets flag for new task */
   do while GRAPHICS_BUFFER (SCREEN_NO - 1).GPR_BUSY = TRUE;
   end;
   GRAPHICS_BUFFER (SCREEN_NO - 1).GPR_BUSY = TRUE;
end GPR_BUSY;
DISPLAY: procedure (SOURCE, SCREEN_NO, DATA_TYPE, DISTANCE) public ;
  declare SOURCE pointer ;
  declare MESSAGE based SOURCE byte ;
  declare SCREEN_NO byte ;
  declare DATA_TYPE byte ;
  declare DISTANCE word ;

  if SCREEN_NO <> 0 then /* report to GDU */
    do ;
      /* Put data into a buffer so that the graphics card can access it */
      do while GRAPHICS_BUFFER (SCREEN_NO - 1) . BUFFER_FLAG = BUSY ;
        end ; /* Wait for buffer to empty */
      GRAPHICS_BUFFER (SCREEN_NO - 1) . BUFFER_FLAG = BUSY ; /* Set wait flag */
      GRAPHICS_BUFFER (SCREEN_NO - 1) . DATA_TYPE = DATA_TYPE ;
      call MOV B (SOURCE , @ GRAPHICS_BUFFER (SCREEN_NO - 1) . GRAPHICS_CONTENTS , (DISTANCE + 1)) ;
      do case SCREEN_NO - 1 ;
        GPR_ADDRESS = 400h ;
        GPR_ADDRESS = 300h ;
        GPR_ADDRESS = 200h ;
        end ;
      call GPR_BUSY (SCREEN_NO ) ; /* Wait for graphics card to finish current task*/
      output (GPR_ADDRESS) = 50 ; /* Tell the graphics card to fetch the buffer */
      end ; /* SCREEN_NO = 0 */
    call WRITE (VDU , SOURCE ) ;
    end DISPLAY ;
$ eject
GENERATE_RAM_PATTERN: procedure (ACTION);

/*****************************
* This procedure first tests for the presence and functionality of the DRAM card before generating a pre-defined address pattern. This pattern is designed in a manner that exercises all the address lines for the first 64 Kbytes.
*******************************************************/

declare ACTION byte;
declare OFFSET word;
declare FOUR_TO_THE_I word;
declare I_TIMES_TWO word;
declare PATTERN word;
declare BYTE_UNDER_TEST byte;
declare VALUE byte;
declare I byte;
declare J byte;

PATTERN = 1;
OFFSET = DRAM_BUFFER_MIN; /* Point to the first location */
DRAM_BUFFER (OFFSET) = ZERO_ONES;

if (DRAM_BUFFER (OFFSET) <> ZERO_ONES) or (TMOUT_INTERRUPT = TRUE) then
    do;
        DRAM_TEST_RESULT = FAILED;
        TMOUT_INTERRUPT = FALSE;
    end;
else
  do;
    DRAM_BUFFER (OFFSET) = ONE_ZEROS ;
  if (DRAM_BUFFER (OFFSET) <> ONE_ZEROS) or (TMOUT_INTERRUPT = TRUE) then
    do;
      DRAM_TEST_RESULT = FAILED;
      TMOUT_INTERRUPT = FALSE;
    end;
  else
    do;
      VALUE = 0; /* Initialise value to be written */
    do I = 0 to 7; /* First write unique values */
      I_TIMES_TWO = shl(I,1);
      FOUR_TO_THE_I = shl (PATTERN, I_TIMES_TWO); /* 4**I = shl (1, 2*I) */
      do J = 1 to 3; /* Generate predefined addresses */
        VALUE = VALUE + 1; /* Each location has a unique value */
        do case J - 1; /* OFFSET = FOUR_TO_THE_I * J */
          OFFSET = FOUR_TO_THE_I;
          OFFSET = FOUR_TO_THE_I + FOUR_TO_THE_I;
          OFFSET = FOUR_TO_THE_I + FOUR_TO_THE_I + FOUR_TO_THE_I;
        end; /* case */
        /* Avoid multiplication when generating addresses, */
        /* as this takes about 70 clock cycles */
      if ACTION = PUT then /* Write values */
        do;
          DRAM_BUFFER (OFFSET) = VALUE ;
          if TMOUT_INTERRUPT = TRUE then
            do;
              DRAM_TEST_RESULT = FAILED;
              TMOUT_INTERRUPT = FALSE;
            end;
          end;
        end;
      else /* ACTION = Read */
        do;
          BYTE_UNDER_TEST = DRAM_BUFFER (OFFSET);
          if BYTE_UNDER_TEST <> VALUE or TMOUT_INTERRUPT = TRUE then
            do;
              DRAM_TEST_RESULT = FAILED;
              TMOUT_INTERRUPT = FALSE;
            end;
          end;
        end; /* do 1 */
      end; /* do J */
    end; /* do I */
  end; /* else */
end; /* else */

end GENERATE_RAM_PATTERN;
POST_DRAM_TEST: procedure (BUS_CHOICE) public;

declare BUS_CHOICE byte;
declare STORE byte;

DRAM_TEST_RESULT = PASSED; /* Be optimistic */
if BUS_CHOICE = MBUS then
  do;
    DRAM_BUFFER_PTR = DDRAM_MBUS_BUFFER_1;
    TEST_RESULT.DRAM_MBUS = PASSED; /* Be optimistic */
    call GENERATE_RAM_PATTERN (PUT);
    if DRAM_TEST_RESULT = PASSED then
      call GENERATE_RAM_PATTERN (GET);
    end;
  end; /* if BUS_CHOICE = MBUS */
else /* BUS_CHOICE = LOCAL BUS */
  do;
    DRAM_BUFFER_PTR = DDRAM_LBUS_BUFFER_1;
    TEST_RESULT.DRAM_LBUS = PASSED; /* Be optimistic */
    call GENERATE_RAM_PATTERN (PUT);
    if DRAM_TEST_RESULT = PASSED then
      call GENERATE_RAM_PATTERN (GET);
    end;
  end; /* if BUS_CHOICE = MBUS */
end POST_DRAM_TEST;
TEST_SLAVE: procedure (SLAVE_ADDRESS, SLAVE_CARD_NO, REPORT_POST_RESULTS);

/* Checks if slave is present, and, if so, issues a self-test command */
/* and then gives the slave a chance to report its test status */

declare SLAVE_ADDRESS word;
declare SLAVE_CARD_NO byte;
declare REPORT_POST_RESULTS byte;
declare GPR_VALUE byte;
declare LOOP_COUNTER byte;

TEST_RESULT.SLAVE(SLAVE_CARD_NO) = input (SLAVE_ADDRESS);
if (TEST_RESULT.SLAVE(SLAVE_CARD_NO) <> PRESENT)
or (TMOUT_INTERRUPT = TRUE) then
do;
    TMOUT_INTERRUPT = FALSE;
    TEST_RESULT.SLAVE(SLAVE_CARD_NO) = ABSENT;
    SLAVE_RESULT.SLAVE_BOARD (SLAVE_CARD_NO) = ABSENT;
end;
else /* issue command to slave to send POST results to global RAM */
do;
    if SLAVE_CARD_NO < 3 then /* the slave card is a graphics card */
do;
        do case SLAVE_CARD_NO;
            GPR_VALUE = 51;
            GPR_VALUE = 52;
            GPR_VALUE = 53;
        end; /* case */
        output (SLAVE_ADDRESS - 1) = GPR_VALUE; /* Initialise graphics card */
        /* buffer pointer */
call TIME (500); /* Wait for graphics card to read GPR register */
end;

output (SLAVE_ADDRESS - 1) = REPORT_POST_RESULTS;

LOOP_COUNTER = 0;
if REPORT_POST_RESULTS = REPORT_TOEPROM_RAM then
   /* The primary processor is running this code,
      so the results will be on the EPROM_RAM card. */
   do;
   do while (MAP_SLAVE_RESULT (SLAVE_CARD_NO) = UNTESTED)
      and (LOOP_COUNTER < MAX_LOOP_COUNTER);
      /* Give the slave a chance to reply */
      call TIME (1000);
      LOOP_COUNTER = LOOP_COUNTER + 1;
   end;
   SLAVE_RESULT.SLAVE_BOARD (SLAVE_CARD_NO)
      = MAP_SLAVE_RESULT (SLAVE_CARD_NO);
   /* Copy the result to an on-board location. */
   end;
else /* TMAP is running this code, so the results will be sent to DRAM */
   do;
   do while (TMAP_SLAVE_RESULT (SLAVE_CARD_NO) = UNTESTED)
      and (LOOP_COUNTER < MAX_LOOP_COUNTER);
      /* Give the slave a chance to reply */
      call TIME (1000);
      LOOP_COUNTER = LOOP_COUNTER + 1;
   end;
   SLAVE_RESULT.SLAVE_BOARD (SLAVE_CARD_NO)
      = TMAP_SLAVE_RESULT (SLAVE_CARD_NO);
   end;
end TEST_SLAVE ;

TEST_TMAP: procedure ;

TMAP_FLAG = INTERROGATE_TMAP;
call TIME (100);
if TMAP_FLAG <> TMAP_PRESENT then
   /* TMAP should have reported its presence by now */
   TMAP_PRESENCE = NOT_PRESENT;
else /* acknowledge TMAP presence */
   do;
      TMAP_PRESENCE = PRESENT;
      TMAP_FLAG = PRESENT;
   end;
end TEST_TMAP;

$ eject
PROTECTED_MODE_POST: procedure;

declare SLAVE_ADDRESS word;
declare REPORTING_ADDRESS word;
declare REPORT_SLAVE_RESULTS byte;
declare SLAVE_CARD_NO byte;
declare COUNTER byte;

declare SYSTEM_FAILURE (*) byte data
(ESC, '1[2J', BEL, BEL, BEL, ESC, '1;OH',
'**** SYSTEM FAILURE HAS OCCURRED !!!!!!', BEL, BEL, BEL, CR, LF, EOM);

declare NO_GRAPHICS_CARD (*) byte data
(ESC, '1[2J', BEL, BEL, BEL, ESC, '15;OH',
'**** CRITICAL ERROR : ', BEL, BEL, BEL, CR, LF, LF,
'SYSTEM HALTED DUE TO NO GRAPHICS CARD PRESENT !!!!!!', CR, LF, EOM);

declare DRAM_CAUSED_HALT (*) byte data
(ESC, '1[2J', BEL, BEL, BEL, ESC, '15;OH',
'**** CRITICAL ERROR : ', BEL, BEL, BEL, CR, LF, LF,
'SYSTEM HALTED DUE TO DRAM FAILURE VIA MULTIBUS !!!!!!', CR, LF, EOM);

declare API_CAUSED_HALT (*) byte data
(ESC, '1[2J', BEL, BEL, BEL, ESC, '15;OH',
'**** CRITICAL ERROR : ', BEL, BEL, BEL, CR, LF, LF,
'SYSTEM HALTED DUE TO API FAILURE !!!!!!', CR, LF, EOM);

/* Power On Self Test */

DRAM_TEST_RESULT = UNTESTED ;
TEST_RESULT.DRAM_MBUS = UNTESTED ;
TEST_RESULT.DRAM_LBUS = UNTESTED ;
TEST_RESULT.SLAVE(0) = UNTESTED ;
TEST_RESULT.SLAVE(1) = UNTESTED ;
TEST_RESULT.SLAVE(2) = UNTESTED ;
TEST_RESULT.SLAVE(3) = UNTESTED ;

TMOUT_INTERRUPT = FALSE ;
TMAP_PRESENCE = FALSE ;

if INITIALISATION_FINISHED = TRUE then
/* System failure has occurred during normal operations */
call WRITE_POLL (VDU, @SYSTEM_FAILURE);
call POST_DRAM_TEST (MBUS);
if TEST_RESULT.DRAM_MBUS <> PASSED then
do;
output (STATUS_LATCH_ADR) = MBUS_DRAM_FAILURE;
output (SLAVE_ADDRESS_1) = MBUS_DRAM_FAILURE;
output (SLAVE_ADDRESS_2) = MBUS_DRAM_FAILURE;
output (SLAVE_ADDRESS_3) = MBUS_DRAM_FAILURE;
if TEST_RESULT.MPSC = PASSED then
call WRITE_POLL (VDU, @DRAM_CAUSED_HALT);
halt;
end;

/* All buffer transfers to slave cards take place via DRAM and Multibus, */
/* and, therefore the DRAM test via Multibus is defined to be critical. */

/* Ideally, if the primary processor and the DRAM connection via the */
/* Multibus has failed (i.e. MAP = FAILED AND DRAM_MBUS = FAILED) then */
/* halt, because the secondary processor running as MAP has no global RAM, */
/* so the slaves are unable to report their status. */

/* Now test for presence of slave cards */
do SLAVE_CARD_NO = 0 to 3;
SLAVE_RESULT.SLAVE_BOARD(SLAVE_CARD_NO) = UNTESTED;
if PROCESSOR_ID = TMAP then /* Initialise the DRAM locations */
TMAP_SLAVE_RESULT (SLAVE_CARD_NO) = UNTESTED;
else /* Initialise the EPROM/RAM locations */
MAP_SLAVE_RESULT (SLAVE_CARD_NO) = UNTESTED;
end; /* do */

if PROCESSOR_ID = TMAP then
/* TMAP has taken over as MAP */
REPORT_SLAVE_RESULTS = REPORT_TO_DRAM;
else /* normal case, i.e. MAP is running as MAP */
REPORT_SLAVE_RESULTS = REPORT_TO_EPROM_RAM;

do SLAVE_CARD_NO = 0 to 3;
do case SLAVE_CARD_NO;
SLAVE_ADDRESS = SLAVE_ADDRESS_1 + 1; /* First graphics card */
SLAVE_ADDRESS = SLAVE_ADDRESS_2 + 1; /* Second graphics card */
SLAVE_ADDRESS = SLAVE_ADDRESS_3 + 1; /* Third graphics card */
SLAVE_ADDRESS = SCMB_ADDRESS + 1; /* SCMB card */
end; /* Case */
call TEST_SLAVE (SLAVE_ADDRESS, SLAVE_CARD_NO, REPORT_SLAVE_RESULTS);
end; /* for loop */
/* Now set up the appropriate graphics screen */

GIMAVAILABLE = FALSE;
SLAVE_CARD_NO = 0;

do while (GIMAVAILABLE = FALSE) and (SLAVE_CARD_NO < 3);
    if (SLAVE_RESULT.SLAVE_BOARD (SLAVE_CARD_NO)) = PASSED then
        /* The first GIM that is detected present and that has passed
           its tests gets the task of reporting the status of the system. */
        do;
            GIMAVAILABLE = SLAVE_CARD_NO + 1;
            do case SLAVE_CARD_NO;
                REPORTING_ADDRESS = SLAVE_ADDRESS_1;  /* GIM 1 */
                REPORTING_ADDRESS = SLAVE_ADDRESS_2;  /* GIM 2 */
                REPORTING_ADDRESS = SLAVE_ADDRESS_3;  /* GIM 3 */
                end;
            end;
        else /* the graphics card is either absent or has failed,
                so try the next one. */
            SLAVE_CARD_NO = SLAVE_CARD_NO + 1;
        end; /* while */
    end;

if GIMAVAILABLE = FALSE then
    do;
        output (STATUS_LATCH_ADR) = NO_GRAPHICS_CARDS;
        if TEST_RESULT.MPSC = PASSED then
            call WRITE_POLL (VDU, NO_GRAPHICS_CARD);
        halt; /* Critical if no graphics cards are present */
    end;
/* If the processor continues execution from here, this implies that */

(a) at least one processor card has passed all critical tests
(b) the EPROM/RAM card is fully operational
(c) the dynamic RAM card is fully operational via Multibus
(d) there is at least one graphics card present in the system.

All status reporting is therefore done via the normal manner of buffer transfer from this point onwards. */

if PROCESSOR_ID <> TMAP then

/* TMAP has taken over as MAP. If this is the case, then the POST will */
/* report an error condition to the graphics card and bypass the local */
/* bus DRAM test. All future run-time references to local bus DRAM are */
/* hence invalid. */

call POST_DRAM_TEST (LBUS);

SCREENS_AVAILABLE = 0;
do COUNTER = 0 to 2;
  if TEST_RESULT.SLAVE(COUNTER) = PRESENT then
    SCREENS_AVAILABLE = SCREENS_AVAILABLE + 1;
  end;
call TIME (10000);
/* TMAP has been delayed in POST for 100 ms, so wait for it to catch up */
call TEST_TMAP;
end PROTECTED_MODE_POST;
eject
PRINT_TEST_RESULTS: procedure (CARD_NO) ;

/* Report all non-critical tests that have failed */

declare CARD_NO byte;
declare GPR_ADDRESS word;
declare SLAVE_CARD_NO byte;

declare COPROCESSOR_FAILURE (*) byte data
   ('NUMERIC COPROCESSOR FAULTY', CR, LF, EOM);

declare MPSC_FAILURE (*) byte data
   ('MPSC FAULTY', CR, LF, EOM);

declare LOCAL_BUS_FAILURE (*) byte data
   ('LOCAL BUS TEST HAS FAILED', CR, LF, EOM);

declare SDB_FAILURE (*) byte data
   ('SDB CARD NOT PRESENT OR FAULTY', CR, LF, EOM);

declare SCMB_FAILURE (*) byte data
   ('SCMB CARD NOT PRESENT OR FAULTY', CR, LF, EOM);

declare LBUS_DRAM_NOT_PRESENT (*) byte data
   ('LOCALBUS DRAM NOT PRESENT OR FAULTY', CR, LF, EOM);

declare GIM_1_NOT_PRESENT (*) byte data
   ('GIM 1 NOT PRESENT', CR, LF, EOM);

declare GIM_2_NOT_PRESENT (*) byte data
   ('GIM 2 NOT PRESENT', CR, LF, EOM);

declare GIM_3_NOT_PRESENT (*) byte data
   ('GIM 3 NOT PRESENT', CR, LF, EOM);

declare GIM_1_INT_PENDING (*) byte data
   ('GIM 1 HAS AN INTERRUPT PENDING', CR, LF, EOM);

declare GIM_2_INT_PENDING (*) byte data
   ('GIM 2 HAS AN INTERRUPT PENDING', CR, LF, EOM);

declare GIM_3_INT_PENDING (*) byte data
   ('GIM 3 HAS AN INTERRUPT PENDING', CR, LF, EOM);

declare GIM_1_ERROR (*) byte data
   ('GIM 1 IS REPORTING AN ERROR CODE', CR, LF, EOM);
declare GIM_2_ERROR (*) byte data
   ('GIM 2 IS REPORTING AN ERROR CODE',
    CR, LF, EOM);

declare GIM_3_ERROR (*) byte data
   ('GIM 3 IS REPORTING AN ERROR CODE',
    CR, LF, EOM);

declare GIM_1_CPU_FAILURE (*) byte data
   ('GIM 1 HAS REPORTED A CPU FAILURE',
    CR, LF, EOM);

declare GIM_2_CPU_FAILURE (*) byte data
   ('GIM 2 HAS REPORTED A CPU FAILURE',
    CR, LF, EOM);

declare GIM_3_CPU_FAILURE (*) byte data
   ('GIM 3 HAS REPORTED A CPU FAILURE',
    CR, LF, EOM);

declare SCMB_CPU_FAILURE (*) byte data
   ('SCMB CARD HAS REPORTED A CPU FAILURE',
    CR, LF, EOM);

declare GIM_1_RAM_FAILURE (*) byte data
   ('GIM 1 HAS REPORTED A RAM FAILURE',
    CR, LF, EOM);

declare GIM_2_RAM_FAILURE (*) byte data
   ('GIM 2 HAS REPORTED A RAM FAILURE',
    CR, LF, EOM);

declare GIM_3_RAM_FAILURE (*) byte data
   ('GIM 3 HAS REPORTED A RAM FAILURE',
    CR, LF, EOM);

declare SCMB_RAM_FAILURE (*) byte data
   ('SCMB CARD HAS REPORTED A RAM FAILURE',
    CR, LF, EOM);

declare GIM_1_REPORT_FAILURE (*) byte data
   ('GIM 1 HAS FAILED TO REPORT POST RESULTS',
    CR, LF, EOM);

declare GIM_2_REPORT_FAILURE (*) byte data
   ('GIM 2 HAS FAILED TO REPORT POST RESULTS',
    CR, LF, EOM);
declare GIM_3_REPORT_FAILURE (*) byte data
('GIM 3 HAS FAILED TO REPORT POST RESULTS',
CR, LF, EOM);

declare SCMB_REPORT_FAILURE (*) byte data
('SCMB CARD HAS FAILED TO REPORT POST RESULTS',
CR, LF, EOM);

declare MAP_FAILURE (*) byte data
('TMAP HAS TAKEN OVER AS MAP DUE TO MAP FAILURE...', CR, LF,
'ALL DRAM LOCAL BUS ACCESSES ARE NOW INVALID!!!', CR, LF, EOM);

declare TMAP_NOT_PRESENT (*) byte data
('TMAP CPU CARD NOT PRESENT', CR, LF, EOM);

declare TMAP_FAULTY (*) byte data
('TMAP CPU CARD FAULTY', CR, LF, EOM);

declare EMAC_FAULTY (*) byte data
('EMAC CARD FAULTY OR NOT PRESENT', CR, LF, EOM);

declare SWITCH_1_FAULTY (*) byte data
('SWITCH 1 DETECTED CLOSED', CR, LF, EOM);

declare SWITCH_2_FAULTY (*) byte data
('SWITCH 2 DETECTED CLOSED', CR, LF, EOM);

declare SWITCH_3_FAULTY (*) byte data
('SWITCH 3 DETECTED CLOSED', CR, LF, EOM);

declare SWITCH_4_FAULTY (*) byte data
('SWITCH 4 DETECTED CLOSED', CR, LF, EOM);

declare SWITCH_5_FAULTY (*) byte data
('SWITCH 5 DETECTED CLOSED', CR, LF, EOM);

declare SWITCH_6_FAULTY (*) byte data
('SWITCH 6 DETECTED CLOSED', CR, LF, EOM);

declare SWITCH_7_FAULTY (*) byte data
('SWITCH 7 DETECTED CLOSED', CR, LF, EOM);

declare KEY_STUCK (*) byte data
('KEY DETECTED STUCK : ', CR, LF, EOM);

declare SKM1_STUCK (*) byte data
('SKM1 ', EOM);

declare SKM2_STUCK (*) byte data
('SKM2 ', EOM);
declare SKM3_STUCK (*) byte data
('SKM3 1, EOM);

declare QKBM_STUCK (*) byte data
('QKBM 1, EOM);

declare SW1_STUCK (*) byte data
('SWITCH 1', CR, LF, EOM);

declare SW2_STUCK (*) byte data
('SWITCH 2', CR, LF, EOM);

declare SW3_STUCK (*) byte data
('SWITCH 3', CR, LF, EOM);

declare SW4_STUCK (*) byte data
('SWITCH 4', CR, LF, EOM);

declare SW5_STUCK (*) byte data
('SWITCH 5', CR, LF, EOM);

declare SW6_STUCK (*) byte data
('SWITCH 6', CR, LF, EOM);

declare SW7_STUCK (*) byte data
('SWITCH 7', CR, LF, EOM);

declare SW8_STUCK (*) byte data
('SWITCH 8', CR, LF, EOM);

declare SECTION1_STUCK (*) byte data
('SECTION 1', CR, LF, EOM);

declare SECTION2_STUCK (*) byte data
('SECTION 2', CR, LF, EOM);

declare SECTION3_STUCK (*) byte data
('SECTION 3', CR, LF, EOM);

declare SECTION4_STUCK (*) byte data
('SECTION 4', CR, LF, EOM);
call DISPLAY (GERASE_SCREEN, GIMAVAILABLE, 
TEXT, LENGTH(ERASE_SCREEN));

if TEST_RESULT.COPROCESSOR <> PASSED then 
call DISPLAY (@COPROCESSOR_FAILURE, GIMAVAILABLE, 
TEXT, LENGTH (COPROCESSOR_FAILURE));

if TEST_RESULT.MPSC <> PASSED then 
call DISPLAY (@MPSC_FAILURE, GIMAVAILABLE, 
TEXT, LENGTH (MPSC_FAILURE));

if TEST_RESULT.LOCAL_BUS <> PASSED then 
call DISPLAY (@LOCAL_BUS_FAILURE, GIMAVAILABLE, 
TEXT, LENGTH (LOCAL_BUS_FAILURE));

/*/ Display SDB card test results */

if TEST_RESULT.SDB_BOARD <> PASSED then 
call DISPLAY (@SDB_FAILURE, GIMAVAILABLE, 
TEXT, LENGTH (SDB_FAILURE));

/*/ Display DRAM test results */

if TEST_RESULT.DRAM_LBUS <> PASSED then 
call DISPLAY (@LBUS_DRAM_NOT_PRESENT, GIMAVAILABLE, 
TEXT, LENGTH (LBUS_DRAM_NOT_PRESENT));

if PROCESSOR_ID = TMAP then /* TMAP has taken over as MAP */
call DISPLAY 
(@MAP_FAILURE, GIMAVAILABLE, TEXT, LENGTH(MAP_FAILURE)) ;
else 
do;
if TMAP_PRESENCE = TRUE then 
do;
if TMAP_RESULT.TMAP_BOARD <> PASSED then 
call DISPLAY (@TMAP_FAULTY, GIMAVAILABLE, 
TEXT, LENGTH (TMAP_FAULTY));
end;
else 
call DISPLAY (@TMAP_NOT_PRESENT, GIMAVAILABLE, 
TEXT, LENGTH (TMAP_NOT_PRESENT));
end;
do SLAVE_CARD_NO = 0 to 3;
  if TEST_RESULT.SLAVE(SLAVE_CARD_NO) <> PRESENT then do;
    if TEST_RESULT.SLAVE(SLAVE_CARD_NO) = ABSENT then do;
      do case SLAVE_CARD_NO;
        call DISPLAY (@GIM_1_NOT_PRESENT, GIM_AVAILABLE, TEXT, LENGTH (GIM_1_NOT_PRESENT)); /* 0 */
        call DISPLAY (@GIM_2_NOT_PRESENT, GIM_AVAILABLE, TEXT, LENGTH (GIM_2_NOT_PRESENT)); /* 1 */
        call DISPLAY (@GIM_3_NOT_PRESENT, GIM_AVAILABLE, TEXT, LENGTH (GIM_3_NOT_PRESENT)); /* 2 */
        call DISPLAY (@SCMB_FAILURE, GIM_AVAILABLE, TEXT, LENGTH (SCMB_FAILURE)); /* 3 */
      end;
    end;
  end;
else /* GIM is in incorrect status */
  do;
    if (TEST_RESULT.SLAVE(SLAVE_CARD_NO) and INT_PENDING_MASK) = TRUE then do;
      do case SLAVE_CARD_NO;
        call DISPLAY (@GIM_1_INT_PENDING, GIM_AVAILABLE, TEXT, LENGTH (GIM_1_INT_PENDING)); /* 0 */
        call DISPLAY (@GIM_2_INT_PENDING, GIM_AVAILABLE, TEXT, LENGTH (GIM_2_INT_PENDING)); /* 1 */
        call DISPLAY (@GIM_3_INT_PENDING, GIM_AVAILABLE, TEXT, LENGTH (GIM_3_INT_PENDING)); /* 2 */
        /* do nothing for the SCMB board */
      end;
    end;
    if (TEST_RESULT.SLAVE(SLAVE_CARD_NO) and GIM_ERROR_MASK) = TRUE then do;
      do case SLAVE_CARD_NO;
        call DISPLAY (@GIM_1_ERROR, GIM_AVAILABLE, TEXT, LENGTH (GIM_1_ERROR)); /* 0 */
        call DISPLAY (@GIM_2_ERROR, GIM_AVAILABLE, TEXT, LENGTH (GIM_2_ERROR)); /* 1 */
        call DISPLAY (@GIM_3_ERROR, GIM_AVAILABLE, TEXT, LENGTH (GIM_3_ERROR)); /* 2 */
        /* do nothing for the SCMB board */
      end; /* case */
      /* if GIM_ERROR */
      end; /* else GIM in incorrect status */
    end; /* if TEST_RESULT.SLAVE(SLAVE_CARD_NO) <> PRESENT */
else /* retrieve slave self-test results from global RAM */
  do;
    if SLAVE_RESULT.SLAVE_BOARD (SLAVE_CARD_NO) <> PASSED then
      do;
        if SLAVE_RESULT.SLAVE_BOARD (SLAVE_CARD_NO) = CPU_FAILURE then
          do;
            do case SLAVE_CARD_NO;
              call DISPLAY (GGIM_1_CPU_FAILURE, GIM_AVAILABLE,
              TEXT, LENGTH (GGIM_1_CPU_FAILURE)); /* 0 */
              call DISPLAY (GGIM_2_CPU_FAILURE, GIM_AVAILABLE,
              TEXT, LENGTH (GGIM_2_CPU_FAILURE)); /* 1 */
              call DISPLAY (GGIM_3_CPU_FAILURE, GIM_AVAILABLE,
              TEXT, LENGTH (GGIM_3_CPU_FAILURE)); /* 2 */
              call DISPLAY (GSCMB_CPU_FAILURE, GIM_AVAILABLE,
              TEXT, LENGTH (GSCMB_CPU_FAILURE)); /* 3 */
            end; /* case */
          end;
        else /* slave has failed to report a valid status */
          do;
            do case SLAVE_CARD_NO;
              call DISPLAY (GGIM_1_REPORT_FAILURE, GIM_AVAILABLE,
              TEXT, LENGTH (GGIM_1_REPORT_FAILURE)); /* 0 */
              call DISPLAY (GGIM_2_REPORT_FAILURE, GIM_AVAILABLE,
              TEXT, LENGTH (GGIM_2_REPORT_FAILURE)); /* 1 */
              call DISPLAY (GGIM_3_REPORT_FAILURE, GIM_AVAILABLE,
              TEXT, LENGTH (GGIM_3_REPORT_FAILURE)); /* 2 */
              call DISPLAY (GSCMB_REPORT_FAILURE, GIM_AVAILABLE,
              TEXT, LENGTH (GSCMB_REPORT_FAILURE)); /* 3 */
            end; /* case */
          end; /* else */
        end; /* else */
      end; /* SLAVE_RESULT.SLAVE_BOARD(SLAVE_CARD_NO) <> PASSED */
    end; /* else */
  end; /* do SLAVE_CARD_NO = 0 to 2 */
if TEST_RESULT.API_EMAC <> PASSED then
    call DISPLAY (EMACFAULTY, GIM_AVAILABLE,
                  TEXT, LENGTH (EMAC_FAULTY));

else /* EMAC switch results are valid */
    do;
        if TEST_RESULT.EMAC_SWITCH (1) <> PASSED then
            call DISPLAY (SWITCH_1_FAULTY, GIM_AVAILABLE,
                          TEXT, LENGTH (SWITCH_1_FAULTY));

        if TEST_RESULT.EMAC_SWITCH (2) <> PASSED then
            call DISPLAY (SWITCH_2_FAULTY, GIM_AVAILABLE,
                          TEXT, LENGTH (SWITCH_2_FAULTY));

        if TEST_RESULT.EMAC_SWITCH (3) <> PASSED then
            call DISPLAY (SWITCH_3_FAULTY, GIM_AVAILABLE,
                          TEXT, LENGTH (SWITCH_3_FAULTY));

        if TEST_RESULT.EMAC_SWITCH (4) <> PASSED then
            call DISPLAY (SWITCH_4_FAULTY, GIM_AVAILABLE,
                          TEXT, LENGTH (SWITCH_4_FAULTY));

        if TEST_RESULT.EMAC_SWITCH (5) <> PASSED then
            call DISPLAY (SWITCH_5_FAULTY, GIM_AVAILABLE,
                          TEXT, LENGTH (SWITCH_5_FAULTY));

        if TEST_RESULT.EMAC_SWITCH (6) <> PASSED then
            call DISPLAY (SWITCH_6_FAULTY, GIM_AVAILABLE,
                          TEXT, LENGTH (SWITCH_6_FAULTY));

        if TEST_RESULT.EMAC_SWITCH (7) <> PASSED then
            call DISPLAY (SWITCH_7_FAULTY, GIM_AVAILABLE,
                          TEXT, LENGTH (SWITCH_7_FAULTY));
if (TEST_RESULT.STUCK_MODULE <> PASSED)
and (TEST_RESULT.STUCK_KEY <> PASSED) then
    do;
    /* This is determined in the real mode applications code.
    Both variables would have been initialised to PASSED if no
    stuck key was detected during power-up. If this is not the case,
    then it remains to be established which key is causing the problem. */
    if (TEST_RESULT.STUCK_MODULE >= 0)
and (TEST_RESULT.STUCK_MODULE <= 8)
and (TEST_RESULT.STUCK_KEY >= 0)
and (TEST_RESULT.STUCK_KEY <= 1) then
        /* Valid stuck key status */
        do;

        call DISPLAY (QKEY_STUCK, GIM_AVAILABLE, TEXT, LENGTH (KEY_STUCK));

        do case TEST_RESULT.STUCK_MODULE;
        call DISPLAY (QSKM1_STUCK, GIM_AVAILABLE, TEXT, LENGTH (SKM1_STUCK));
        ; /* 1 */
        call DISPLAY (QSKM2_STUCK, GIM_AVAILABLE, TEXT, LENGTH (SKM2_STUCK));
        ; /* 3 */
        call DISPLAY (QSKM3_STUCK, GIM_AVAILABLE, TEXT, LENGTH (SKM3_STUCK));
        ; /* 5 */
        ; /* 6 */
        ; /* 7 */
        call DISPLAY (QKBM_STUCK, GIM_AVAILABLE, TEXT, LENGTH (QKBM_STUCK));
        end; /* case STUCK_MODULE */
do case TEST_RESULT.STUCK_KEY;
  do; /* 0 */
    if TEST_RESULT.STUCK_MODULE = 8 then /* QKBM stuck */
      call DISPLAY (@SECTION1_STUCK, GIM_AVAILABLE, TEXT, LENGTH (SECTION1_STUCK));
    else
      call DISPLAY (@SW1_STUCK, GIM_AVAILABLE, TEXT, LENGTH (SW1_STUCK));
    end; /* 0 */
  do; /* 1 */
    if TEST_RESULT.STUCK_MODULE = 8 then /* QKBM stuck */
      call DISPLAY (@SECTION2_STUCK, GIM_AVAILABLE, TEXT, LENGTH (SECTION2_STUCK));
    else
      call DISPLAY (@SW2_STUCK, GIM_AVAILABLE, TEXT, LENGTH (SW2_STUCK));
    end; /* 1 */
  do; /* 2 */
    if TEST_RESULT.STUCK_MODULE = 8 then /* QKBM stuck */
      call DISPLAY (@SECTION3_STUCK, GIM_AVAILABLE, TEXT, LENGTH (SECTION3_STUCK));
    else
      call DISPLAY (@SW3_STUCK, GIM_AVAILABLE, TEXT, LENGTH (SW3_STUCK));
    end; /* 2 */
  do; /* 3 */
    if TEST_RESULT.STUCK_MODULE = 8 then /* QKBM stuck */
      call DISPLAY (@SECTION4_STUCK, GIM_AVAILABLE, TEXT, LENGTH (SECTION4_STUCK));
    else
      call DISPLAY (@SW4_STUCK, GIM_AVAILABLE, TEXT, LENGTH (SW4_STUCK));
    end; /* 3 */
call DISPLAY (@SW5_STUCK, GIM_AVAILABLE, TEXT, LENGTH (SW5_STUCK));
call DISPLAY (@SW6_STUCK, GIM_AVAILABLE, TEXT, LENGTH (SW6_STUCK));
call DISPLAY (@SW7_STUCK, GIM_AVAILABLE, TEXT, LENGTH (SW7_STUCK));
call DISPLAY (@SW8_STUCK, GIM_AVAILABLE, TEXT, LENGTH (SW8_STUCK));
end; /* case STUCK_KEY */
end; /* Valid stuck key status */
end; /* STUCK_MODULE and STUCK_KEY <> PASSED */
end; /* else EMAC = PASSED */

end PRINT_TEST_RESULTS;

$ eject

INITIALISE_MIDS: procedure;

declare DUMMY byte;

MIDS_START_INDEX   = 0;
MIDS_END_INDEX     = 0;

end INITIALISE_MIDS;

$ eject
PM_INITIALISATION: procedure public;

declare SYSTEM_FAILURE (*) byte data
    (ESC, '^[2J', 'SYSTEM FAILURE has occurred during normal operations',
    CR, LF, LF, 'Power On Self Test has been invoked', EOM);

declare CONTINUE (*) byte data
    (ESC, '[1$DH', 'Strike any key to continue...', EOM);

declare COUNTER byte;
declare LETTER byte;

enable;
output(PIC_MASTER_8259A_ADR2) = UNMASK_SLAVE
    and UNMASK_MPSC
    and UNMASK_TMOUT ;

if INITIALISATION_FINISHED = FALSE then
    BIT_INVOKED = FALSE; /* Initialise flag */

    TEST = TRUE;

call PROTECTED_MODE_POST ;

output (STATUS_LATCH_ADR) = RESET_LATCH;

call DISPLAY (@ERASE_SCREEN, GIM_AVAILABLE,
    TEXT, LENGTH (ERASE_SCREEN));

call PRINT_TEST_RESULTS (GIM_AVAILABLE);

TMAP_SCREEN = 0;
if SCREENS_AVAILABLE > 1 then /* Set up TMAP_SCREEN */
    do
        do case (GIM_AVAILABLE mod 3);
        do;
            if TEST_RESULT.SLAVE(0) = PASSED then
                TMAP_SCREEN = 1;
            else
                TMAP_SCREEN = 2;
            end;
        end;
        if TEST_RESULT.SLAVE(1) = PASSED then
            TMAP_SCREEN = 2;
        else
            TMAP_SCREEN = 3;
        end;
    end;
do;
  if TEST_RESULT.SLAVE(2) = PASSED then
    TMAP_SCREEN = 3;
  else
    TMAP_SCREEN = 0;
  end;
end; /* case */
end; /* if SCREENS_AVAILABLE > 1 */

if INITIALISATION_FINISHED = TRUE then

  /* System failure has occurred during normal operations ! */

  if (GIM_AVAILABLE <> 0) and (TEST_RESULT.RAM = PASSED) then

    /* OK to write to graphics card */
    call DISPLAY
      (SYSTEM_FAILURE, GIM_AVAILABLE, TEXT, LENGTH(SYSTEM_FAILURE));

    call INITIALISE_MIDS;

    output(PIC_SLAVE_8259A_ADR2) = UNMASK_SCBM ;

    INITIALISATION_FINISHED = TRUE;
  TEST = FALSE ;

  if (TEST_RESULT.CPU_BOARD <> PASSED) or
    (TEST_RESULT.SDB_BOARD <> PASSED) or
    (TEST_RESULT.API_BOARD <> PASSED) or
    (TEST_RESULT.DRAM_MBUS <> PASSED) or
    (TEST_RESULT.DRAM_LBUS <> PASSED) or
    (TEST_RESULT.SLAVE(0) <> PASSED) or
    (TEST_RESULT.SLAVE(1) <> PASSED) or
    (TEST_RESULT.SLAVE(2) <> PASSED) or
    (TEST_RESULT.SLAVE(3) <> PASSED) or
    (TMAP_RESULT.TMAP_BOARD <> PASSED) then

    do;
    call DISPLAY (@CONTINUE, GIM_AVAILABLE, TEXT, LENGTH(CONTINUE));
    LETTER = INPUT_CHARACTER (KB);
  end;

end PM_INITIALISATION;

end PROTECTED_MODE_INITIALISATION ;
$ include (PLMPAR.INC)

/******************************************************************************
* Source Filename : PM_INTS.PLM
* Source Compiler : PLM286
* Operating System : DOS 3.10
* Description : Interrupt handlers for interrupts that occur specifically in the protected mode applications.
* Public procedures: REAL_TIME_CLOCK, SCMB
* EPD files : None
* Include files : PLMPAR.INC, PICLITS.INC
*******************************************************************************/

HISTORY Version 1.0 :
    Designed by : P.A. OLANDER    Date : July 1989
    Description : Original

$ eject
$ include (..\STD\PICLITS.INC)
$ eject

declare RESET literally '0'
declare FALSE literally '0'
declare TRUE literally '1'
declare RESET_SCMB_INTERRUPT literally '0'
declare SCMB_ADR literally '500h'
declare EOI_8259A_INT0 literally '60h'
declare EOI_8259A_INT1 literally '61h'

$ eject

declare CLOCK_INTERRUPT byte public;
declare CLOCK_1_INTERRUPT byte external;
declare TICK real public;
declare TOCK real public;
declare SECONDS byte public;
declare TEN_MILLISECONDS byte public;
declare SCMB_INTERRUPT byte public;
declare API_INTERRUPT byte public;

REAL_TIME_CLOCK: procedure interrupt public;

    CLOCK_INTERRUPT = TRUE;
    CLOCK_1_INTERRUPT = TRUE;
    TICK = TICK + 1.0;
    TEN_MILLISECONDS = TEN_MILLISECONDS + 1;
    if (TEN_MILLISECONDS mod 100) = 0 then
        do;
            SECONDS = SECONDS + 1;
            TOCK = TOCK + 1.0;
        end;
    end;

end REAL_TIME_CLOCK;
/* Slave PIC interrupt handlers */

API: procedure interrupt public;

API_INTERRUPT = TRUE;

output (PIC_SLAVE_8259A_ADR1) = E01_8259A_INTC;

end API;

SCMB: procedure interrupt public;

SCMB_INTERRUPT = TRUE;
output (SCMB_ADR + 2) = RESET_SCMB_INTERRUPT;

output (PIC_SLAVE_8259A_ADR1) = E01_8259A_INT1;

end SCMB;

end PROTECTED_MODE_INTERRUPTS;
$ include (PLMPAR.INC)

/*************************************************************************/
* MODULE NAME : PROTECTED_MODE_APPLICATIONS
*************************************************************************/
* Source Filename : PM_APP.PLML
* Source Compiler : PLM286
* Operating System : DOS 3.10
* Description : Applications code to run the protected mode functionality demonstrations.
* Public procedures: MAIN
* EPD files : STDIO.EPD
* STDCONVT.EPD
* STDINIT.EPD
* STDBIT.EPD
* STDCPU.EPD
* STDRAM.EPD
* STDERAM.EPD
* PM_SEG.SEPD
* PM_INIT.EPD
* Include files : PLMPAR.INC
* LITS.INC
* PICLITS.INC

*************************************************************************/

*************************************************************************/
* HISTORY Version 1.0 :
* Designed by : P.A. OLander Date : September 1989
* Description : Original

*************************************************************************/
PROTECTED_MODE_APPLICATIONS: do;

$ include (..\STD\STDIO.EPD)
$ eject
$ include (..\STD\LITS.INC)
$ eject
$ include (..\STD\STDCONVT.EPD)
$ eject
$ include (..\STD\STDINIT.EPD)
$ eject
$ include (..\STD\STDBIT.EPD)
$ eject
$ include (..\STD\STDCPU.EPD)
$ eject
$ include (..\STD\STDRAM.EPD)
$ eject
$ include (..\STD\STDERAM.EPD)
$ eject
$ include (..\STD\PCLITS.INC)
$ eject
$ include (PM_SEGS.EPD)
$ eject
$ include (PM_INIT.EPD)
$ eject
$ include (PM_INTS.EPD)
$ eject

declare NOT_BUSY literally '0';
declare TEXT literally '1';
declare FILL literally '2';
declare MAX_ASCII literally '127';
declare KEYPRESS literally '2';
declare JOYSTICK literally '3';
declare SET literally '11';
declare EXIT literally '2';
declare SHOW_COUNT literally '3';
declare DISPLAY_INTERVAL literally '5';
declare TMAP_TEST_1 literally '11';
declare TMAP_TEST_2 literally '2';
declare MBUS literally '05h';
declare LBUS literally '0ah';
declare SOFTKEY_1 literally '11';
declare SOFTKEY_2 literally '12';
declare MAX_COUNT literally '10';
declare ANY_LETTER literally '127';
declare INVALID_CHOICE literally '30';
declare INVOKE_SELF_TEST literally '5';
declare INVOKE_EPROM_RAM_TEST literally '6';
declare INVOKE_DRAM_TEST literally '7';
declare INVOKE_SLAVE_TEST literally '8';
declare INVOKE_API_TEST literally '12';
declare INVOKE_SDB_TEST literally '13';
declare ISSUE_API_SELF_TEST_COMMAND literally '40h';
declare API_SELF_TEST_PASSED literally '37h';
declare SET_TEST_FLAG literally '14';
declare CLEAR_TEST_FLAG literally '15';
declare INVOKE_MSCC_TEST literally '16';
declare SCMB_ADR literally '500h';
declare MSCC_ADR literally '600h';
declare MSCC_STATUS_REGISTER literally '601h';
declare TEST_MSCC literally '0fah';
declare PERFORM_CPU_TEST literally '4';
declare PERFORM_RAM_TEST literally '5';
declare REPORT_POST_RESULTS literally '0e0h';
declare STATUS_LATCH_ADR literally '0aah';

declare SELECTION_PROMPT (*) byte data
('Select a letter... ESC to quit , EOM);
declare ERASE_SCREEN (*) byte data (ESC, '2J', ESC, '1;OH', EOM);
/* Erases the screen and homes the cursor */
declare POSITION_CURSOR (*) byte data
(ESC, '24;60H', EOM);
declare DELIMITER (*) byte data (' ', EOM);

declare BUSY_4 (*) byte data
(ESC, '4;35H', 'BUSY TESTING', EOM);

declare PASSED_3 (*) byte data
(ESC, '3;35H', 'PASSED', EOM);
declare FAILED_3 (*) byte data
(ESC, '3;3SH', 'FAILED', EOM);

declare PASSED_4 (*) byte data
(ESC, '4;35H', 'PASSED', EOM);
declare FAILED_4 (*) byte data
(ESC, '4;3SH', 'FAILED', EOM);

declare PASSED_5 (*) byte data
(ESC, '5;35H', 'PASSED', EOM);
declare FAILED_5 (*) byte data
(ESC, '5;3SH', 'FAILED', EOM);
declare PASSED_6 (*) byte data
(ESC, "[6;35H", 'PASSED', EOM);

declare FAILED_6 (*) byte data
(ESC, "[6;35H", 'FAILED', EOM);

declare NOT_PRESENT_6 (*) byte data
(ESC, "[6;35H", 'NOT PRESENT', EOM);

declare PASSED_7 (*) byte data
(ESC, "[7;35H", 'PASSED', EOM);

declare FAILED_7 (*) byte data
(ESC, "[7;35H", 'FAILED', EOM);

declare NOT_PRESENT_7 (*) byte data
(ESC, "[7;35H", 'NOT PRESENT', EOM);

declare PASSED_8 (*) byte data
(ESC, "[8;35H", 'PASSED', EOM);

declare FAILED_8 (*) byte data
(ESC, "[8;35H", 'FAILED', EOM);

declare NOT_PRESENT_8 (*) byte data
(ESC, "[8;35H", 'NOT PRESENT', EOM);

declare PASSED_9 (*) byte data
(ESC, "[9;35H", 'PASSED', EOM);

declare FAILED_9 (*) byte data
(ESC, "[9;35H", 'FAILED', EOM);

declare NOT_PRESENT_9 (*) byte data
(ESC, "[9;35H", 'NOT PRESENT', EOM);

declare PASSED_10 (*) byte data
(ESC, "[10;35H", 'PASSED', EOM);

declare FAILED_10 (*) byte data
(ESC, "[10;35H", 'FAILED', EOM);

declare PASSED_11 (*) byte data
(ESC, "[11;35H", 'PASSED', EOM);

declare FAILED_11 (*) byte data
(ESC, "[11;35H", 'FAILED', EOM);

declare PASSED_12 (*) byte data
(ESC, "[12;35H", 'PASSED', EOM);

declare FAILED_12 (*) byte data
(ESC, "[12;35H", 'FAILED', EOM);

declare NOT_PRESENT_13 (*) byte data
(ESC, "[13;35H", 'NOT PRESENT', EOM);

declare KBS_ARRAY (4) byte;
declare ASCII_VALUE (4) byte;
declare GPR_ADDRESS word;
declare TMAP_CPU byte external;
declare TMAP_RAM byte external;
declare TMAP_DRAM byte external;
declare TMAP_PRESENCE byte external;
declare MIDS_START_INDEX byte external;
declare MIDS_END_INDEX byte external;
declare GIM_AVAILABLE byte external;
declare SCREENS_AVAILABLE byte external;
declare BIT_INVOKED byte external;
declare TMOUT_INTERRUPT byte external;
declare CLOCK_1_INTERRUPT byte external;
declare CLOCK_2_INTERRUPT byte external;

/* GRAPHICS_BUFFER, SCMB_BUFFER and TMAP_FLAG are declared public in module PM_SEGS.ASM */

declare K integer;
declare CURRENT_SETTING byte;
declare LETTER byte;
declare SELECTION byte;

% eject

READ_CHAR: procedure;

    LETTER = INPUT_CHARACTER (KB);

end READ_CHAR;

GET_INPUT: procedure (MAX_LETTER);

    declare MAX_LETTER byte;

    call READ_CHAR;

    if ( (LETTER >= 'a') and (LETTER <= 'z') ) then /* convert to upper case */
        LETTER = LETTER - 32;
    if ( (LETTER >= 'A') and (LETTER <= MAX_LETTER) ) then
        SELECTION = LETTER - 'A'; /* keyboard selections */
    else
        if ( (LETTER >= 0) and (LETTER <= (MAX_LETTER - 'A') )
            or (LETTER = ESC) ) then
            SELECTION = LETTER; /* softkey selections */
        else
            SELECTION = INVALID_CHOICE;
    end if;

end GET_INPUT;
LEGAL_INPUT: procedure (MIN_LETTER, MAX_LETTER) byte;

declare MIN_LETTER byte;
declare MAX_LETTER byte;
declare LEGAL_INPUT byte;

LEGAL_INPUT = FALSE;

if (LETTER = ESC)
or (LETTER >= 0) and (LETTER <= 7) /* Allowable softkey inputs */
or (LETTER >= MIN_LETTER) and (LETTER <= MAX_LETTER) ) then
    LEGAL_INPUT = TRUE;

return LEGAL_INPUT;
end LEGAL_INPUT;

DISPLAY_GRAPHICS: procedure (SCREEN_NO, PATTERN_NO);
declare SCREEN_NO byte;
declare PATTERN_NO byte;
declare GPR_VALUE byte;
declare GPR_ADDRESS word;

do case SCREEN_NO - 1;
do;
    GPR_ADDRESS = 400h; /* 0 */
    GPR_VALUE = PATTERN_NO;
    end;
do;
    GPR_ADDRESS = 300h; /* 1 */
    GPR_VALUE = PATTERN_NO;
    end;
do;
    GPR_ADDRESS = 200h; /* 2 */
    GPR_VALUE = PATTERN_NO;
    end;
end;
call GPR_BUSY(SCREEN_NO);
output(GPR_ADDRESS) = GPR_VALUE;
end DISPLAY_GRAPHICS;

GPR_BUSY: procedure (SCREEN_NO); /* Procedure waits for specified */
declare SCREEN_NO byte; /* graphics card to finish task */
/* then sets flag for new task */
do while GRAPHICS_BUFFER (SCREEN_NO - 1).GPR_BUSY = TRUE;
end;
GRAPHICS_BUFFER (SCREEN_NO - 1).GPR_BUSY = TRUE;
end GPR_BUSY;
ENABLE_INTERRUPTS: procedure;

output (PIC_MASTER_8259A_ADR2) = UNMASK_MFSC
and UNMASK_THOUT
and UNMASK_SLAVE  
output (PIC_SLAVE_8259A_ADR2) = UNMASK_SCMB  

end ENABLE_INTERRUPTS;

TEST_API_RESPONSE: procedure;

declare COUNT byte;

output (PIC_SLAVE_8259A_ADR2) = UNMASK_API  
API_INTERRUPT = FALSE;
COUNT = 0;
AP_WR_DATA = ISSUE_API_SELF_TEST_COMMAND;

do while (API_INTERRUPT = FALSE) and (COUNT < MAX_COUNT);
COUNT = COUNT + 1;
call TIME (10000);
end;

if (API_INTERRUPT = TRUE) then
do;
TEST_RESULT.API_BOARD = PASSED;
API_INTERRUPT = FALSE;
end;
else
TEST_RESULT.API_BOARD = FAILED;

output (PIC_SLAVE_8259A_ADR2) = UNMASK_SCMB  
/* Put slave PIC back into default state */

end TEST_API_RESPONSE;

$ eject
TEST_GRAPHICS: procedure;

declare GRAPHICS_MENU (*) byte data
(ESC, '1Z', ESC, '1;OH',
 'Graphics display unit demonstration menu', CR, LF,
 'A Initialize graphics card', CR, LF,
 'B Generate MIDS test pattern', CR, LF,
 'C Generate colour bars', CR, LF,
 'D Generate circles', CR, LF,
 'E Generate softkey pattern', CR, LF,
 'F Disable test message generation', CR, LF,
 'G Change selection of GDU', CR, LF,
 CR, LF,
 CR, LF,
 CR, LF,
 CR, LF,
 CR, LF,
 LF, EOM);

declare GRAPHICS_MSG_1 (*) byte data
(ESC, '1Z', ESC, '1;OH',
 'Graphics display unit selection menu', CR, LF,
 'A Select GDU 1', CR, LF,
 'B Select GDU 2', CR, LF,
 'C Select GDU 3', CR, LF,
 LF, EOM);

declare NOT_AVAILABLE (*) byte data
(ESC, '18;OH',
 'Screen not available --- choose another...', EOM);

declare CONTINUE (*) byte data
(ESC, '20;OH', 'Strike a key to continue...', EOM);

declare GO_ON (*) byte data
(ESC, '1Z', ESC, '120;OH', 'Strike a key to continue...', EOM);

declare SCREEN_NO byte;
declare PATTERN_NO byte;

call DISPLAY (@GRAPHICS_MENU, GIM_AVAILABLE, "EXT", length(GRAPHICS_MENU));
call DISPLAY
(SeleCITION_PROMPT, GIM_AVAILABLE, TEXT, length(SEI_ECTION_PROMPT));
call GET_INPUT ('G');
do while (not LEGAL_INPUT ('A', IGI);
call GET_INPUT ('G');
end;
do while SELECTION <> ESC;

if (SELECTION >= 1) and (SELECTION <= 4) then /* Test pattern */

do;
PATTERN_NO = (SELECTION + 33);
/* GPR values for test pattern numbers are defined to be 34 to 37 */
call DISPLAY_GRAPHICS (SCREEN_NO, PATTERN_NO);
call WRITE (VDU, @CONTINUE);
if (SCREEN_NO = GIM_AVAILABLE) and (SCREENS_AVAILABLE > 1) then
  if SLAVE_RESULT.SLAVE_BOARD(GIM_AVAILABLE mod 3) = PASSED then
    call DISPLAY ((GO_ON, (GIM_AVAILABLE mod 3) + 1, TEXT, length(GO_ON));
  else
    call DISPLAY ((GO_ON, (GIM_AVAILABLE mod 3) + 2, TEXT, length(GO_ON));
  if (SCREEN_NO <> GIM_AVAILABLE) or (SCREENS_AVAILABLE = 1) then
    call DISPLAY ((GCONTINUE, GIM_AVAILABLE, TEXT, length(CONTINUE));
call READ_CHAR;
end;

else /* not a test pattern */
do;
do case SELECTION;
do;
do case GIM_AVAILABLE - 1;
  GPR_ADDRESS = 400h;
  GPR_ADDRESS = 300h;
  GPR_ADDRESS = 200h;
end; /* case */
/* Now send the initialisation command */
output (GPR_ADDRESS) = GIM_AVAILABLE + 50;
end;

; /* B */
;
; /* C */
;
; /* D */
;
; /* E */

GIM_AVAILABLE = FALSE; /* F */
do; /* G */
call DISPLAY
   (@GRAPHICS_MSG_1, GIM_AVAILABLE, TEXT,
    length(GRAPHICS_MSG_1));
call DISPLAY
   (@SELECTION_PROMPT, GIM_AVAILABLE, TEXT,
    length(SELECTION_PROMPT));
call GET_INPUT ('C');

do while (not LEGAL_INPUT ('A', 'C'));
call GET_INPUT ('C');
end;

do while SELECTION <> ESC;
   if SLAVE_RESULT.SLAVE_BOARD((SELECTION + 1) mod 3) = PASSED then
      GIM_AVAILABLE = SELECTION + 1;
   else
      call DISPLAY
         (@NOT_AVAILABLE, GIM_AVAILABLE, TEXT,
          length(NOT_AVAILABLE));
   end;
end;

end; /* case SELECTION */
end; /* not a test pattern */
end; /* SELECTION <> ESC */

end TEST_GRAPHICS;

$ eject
TEST_TMAP: procedure;

declare TMAP_MSG (*) byte data
    (ESC, '[2J', ESC, '[1;OH',
     'BUS MASTER CONTENTION DEMONSTRATION', CR, LF,
     ESC, '[15;OH',
     'Strike a key to continue...', CR, LF, EOM);

declare MAP_MSG_1 (*) byte data
    (ESC, '[5;OH', ' ' MAP COUNT = ', CR,EOM);

declare MAP_MSG_2 (*) byte data
    (ESC, '[2J', ESC, '[5;OH', ' ' MAP COUNT = 0000 ', CR,EOM);

declare I integer;
declare ASCII_MAP_ARRAY_1 (45) byte;
declare ASCII_MAP_ARRAY_2 (45) byte;
declare MAP_COUNTER word;
declare EXIT_TEST byte;
declare FLAG_STATUS byte;
declare SCREEN_COUNTER byte;

TMAP_FLAG = RESET;
SCREEN_COUNTER = 0;
MAP_COUNTER = 0;
do I = 0 to 44;
    ASCII_MAP_ARRAY_1 (I) = MAP_MSG_1 (I);
    ASCII_MAP_ARRAY_2 (I) = MAP_MSG_2 (I);
end;
call DISPLAY (@TMAP_MSG, GIM_AVAILABLE, TEXT, length(TMAP_MSG));

EXIT_TEST = FALSE;
MIDS_START_INDEX = 0;
MIDS_END_INDEX = 0;

do while CLOCK_INTERRUPT = FALSE;
    end;

SECONDS = 0;
do while EXIT_TEST = FALSE;
  if (BUFFER_READY (KB) <> FALSE)
    or (MIDS_START_INDEX <> MIDS_END_INDEX) then
    do;
      EXIT_TEST = TRUE;
      MIDS_START_INDEX = 0;
      MIDS_END_INDEX = 0;
    end;
  end;
  FLAG_STATUS = TMAP_FLAG;
  if FLAG_STATUS = RESET then
    do;
      if EXIT_TEST = TRUE then
        do;
          TMAP_FLAG = EXIT;
          MAP_COUNTER = 0;
        end;
      else;
        do;
          MAP_COUNTER = MAP_COUNTER + 1;
          if SECONDS > 1 then
            do;
              SECONDS = 0;
              call C030_BIN_WORD_TO_ASCII_DEC
                (MAP_COUNTER,GASCII_MAP_ARRAY_2(27));
              call C030_BIN_WORD_TO_ASCII_DEC
                (MAP_COUNTER,GASCII_MAP_ARRAY_1(24));
              if SCREEN_COUNTER = 0 then
                do;
                  TMAP_FLAG = SHOW_COUNT;
                  call DISPLAY
                    (GASCII_MAP_ARRAY_2, GIM_AVAILABLE, TEXT, length(GASCII_MAP_ARRAY_2));
                  call WRITE (VDU,GASCII_MAP_ARRAY_1);
                end;
              MAP_COUNTER = 0;
            do while TMAP_FLAG <> RESET; /* Wait for TMAP to clear flag */
              end;
            TMAP_FLAG = 4; /* Tell the TMAP to reset T_MAP_COUNTER */
            SCREEN_COUNTER = (SCREEN_COUNTER + 1) mod DISPLAY_INTERVAL;
          end;
        else
          do;
            TMAP_FLAG = SET;
          end;
        end; /* else */
      end; /* if EXIT_TEST = TRUE */
  end; /* if FLAG_STATUS = RESET */
end TEST_TMAP;
TEST_SLAVE: procedure (SLAVE_CARD_NO);

declare SLAVE_CARD_NO byte;
declare SLAVE_ADDRESS word;
declare SLAVE_CPU byte;

if SLAVE_CARD_NO >= 0 and (SLAVE_CARD_NO <= 3) then do;
   TEST_RESULT.SLAVE(SLAVE_CARD_NO) = UNTESTED;
   SLAVE_RESULT.SLAVE_BOARD(SLAVE_CARD_NO) = UNTESTED;
   do case SLAVE_CARD_NO;
       SLAVE_ADDRESS = 400h; /* GIM 1 */
       SLAVE_ADDRESS = 300h; /* GIM 2 */
       SLAVE_ADDRESS = 200h; /* GIM 3 */
       SLAVE_ADDRESS = 500h; /* SCMB */
   end; /* case */
   output (SLAVE_ADDRESS) = 7; /* CPU test reports to DRAM */
   call TIME (10000); /* CPU test reports to DRAM */
   do while TMAP_SLAVE_RESULT(SLAVE_CARD_NO) = UNTESTED;
      call TIME (10000); /* MAP has higher priority, so delay */
   end; /* Wait for slave CPU test result */
   SLAVE_CPU = TMAP_SLAVE_RESULT(SLAVE_CARD_NO);

   TMAP_SLAVE_RESULT(SLAVE_CARD_NO) = UNTESTED;
   output (SLAVE_ADDRESS) = 8; /* RAM test reports to DRAM */
   call TIME (10000);
   do while TMAP_SLAVE_RESULT(SLAVE_CARD_NO) = UNTESTED;
      call TIME (10000); /* MAP has higher priority, so delay */
   end; /* Wait for slave CPU test result */
   TEST_RESULT.SLAVE(SLAVE_CARD_NO) = SLAVE_CPU
      and TMAP_SLAVE_RESULT(SLAVE_CARD_NO);
end; /* if SLAVE_CARD_NO is valid */
end TEST_SLAVE;

$ eject
WAIT_FOR_TMAP: procedure;

declare i byte;

do i = 1 to 120;
call TIME (250); /* Delay for 3 seconds */
end;
do while TMAP_FLAG <> RESET;
call TIME (10000); /* 1 second delay */
end;
end WAIT_FOR_TMAP;

$ eject

TMAP_BIT: procedure;

declare TMAP_BIT_MENU (*) byte data
(ESC, '[2J', ESC, '[1;OH',
 'Secondary processor off-line diagnostics menu',
 'Secondary processor self test',
 'Secondary processor EPROM/RAM test',
 'Secondary processor dynamic RAM test (Multibus only)',
 'First graphics interface module test',
 'Second graphics interface module test',
 'Third graphics interface module test',
 'Serial communications to Multibus test',
 'Applications processor interface test',
 'System data bus controller test',
 'Mass storage controller test',
 CR, LF,
 CR, LF,
 CR, LF,
 CR, LF,
 CR, LF,
 CR, LF,
 CR, LF,
 CR, LF,
 CR, LF,
 CR, LF,
 CR, LF,
 CR, LF,
 CR, LF,
 CR, LF,
 LF, EOM);

declare SLAVE_CARD_NO byte;

call DISPLAY (TMAP_BIT_MENU, GIM_AVAILABLE, TEXT, length(TMAP_BIT_MENU));
call DISPLAY (SELECTION_PROMPT, GIM_AVAILABLE, TEXT,
 length(SELECTION_PROMPT));
call GET_INPUT ('J');
do while (not LEGAL_INPUT ('A','J'));
call GET_INPUT ('J');
end;
do while (LETTER <> ESC);
do case SELECTION;

do; /* A */

TMAP_RESULT.TMAP_BOARD = UNTESTED;
TMAP_RESULT.TMAP_CPU = UNTESTED;
TMAP_RESULT.TMAP_ROM = UNTESTED;
TMAP_RESULT.TMAP_MASTER_PIC = UNTESTED;
TMAP_RESULT.TMAP_SLAVE_PIC = UNTESTED;
TMAP_RESULT.TMAP_TMOUT_CIRCUIT = UNTESTED;
TMAP_RESULT.TMAP_PIT = UNTESTED;
TMAP_RESULT.TMAP_COPROCESSOR = UNTESTED;
TMAP_RESULT.TMAP_MPSC = UNTESTED;
TMAP_RESULT.TMAP_RAM = UNTESTED;
TMAP_RESULT.TMAP_ACCESS_TO_1611 = UNTESTED;
TMAP_RESULT.TMAP_LOCAL_BUS = UNTESTED;
TMAP_FLAG = INVOKE_SELF_TEST;
call WAIT_FOR_TMAP;
if TMAP_RESULT.TMAP_BOARD = PASSED then
call DISPLAY (@PASSED_3, GIM_AVAILABLE, TEXT, length(PASSED_3));
else
call DISPLAY (@FAILED_3, GIM_AVAILABLE, TEXT, length(FAILED_3));
end; /* TMAP test */

do; /* B */

TMAP_RESULT.TMAP_ERAM = UNTESTED;
TMAP_RESULT.TMAP_EROM = UNTESTED;
TMAP_FLAG = INVOKE_EPROM_RAM_TEST;
call WAIT_FOR_TMAP;
if TMAP_RESULT.TMAP_ERAM and TMAP_RESULT.TMAP_EROM = PASSED then
call DISPLAY (@PASSED_4, GIM_AVAILABLE, TEXT, length(PASSED_4));
else
call DISPLAY (@FAILED_4, GIM_AVAILABLE, TEXT, length(FAILED_4));
end; /* EPROM/RAM test */

do; /* C */

TMAP_RESULT.TMAP_DRAM_MBUS = UNTESTED;
TMAP_FLAG = INVOKE_DRAM_TEST;
call WAIT_FOR_TMAP;
if TMAP_RESULT.TMAP_DRAM_MBUS = PASSED then
call DISPLAY (@PASSED_5, GIM_AVAILABLE, TEXT, length(PASSED_5));
else
call DISPLAY (@FAILED_5, GIM_AVAILABLE, TEXT, length(FAILED_5));
end; /* DRAM test */
do; /* D */
SLAVE_CARD_NO = 0;
if TEST_RESULT.SLAVE(SLAVE_CARD_NO) = ABSENT then
call DISPLAY (@NOT_PRESENT_6, GIM_AVAILABLE,
 TEXT, length(NOT_PRESENT_6));
else
do;
TMAP_RESULT.TMAP_SLAVE(SLAVE_CARD_NO) = UNTESTED;
TMAP_FLAG = INVOKE_SLAVE_TEST + SLAVE_CARD_NO;
call WAIT_FOR_TMAP;
if TMAP_RESULT.TMAP_SLAVE(SLAVE_CARD_NO) = PASSED then
call DISPLAY (@PASSED_6, GIM_AVAILABLE, TEXT, length(PASSED_6));
else
call DISPLAY (@FAILED_6, GIM_AVAILABLE, TEXT, length(FAILED_6));
end;
end; /* GIM 1 test */
do; /* E */
SLAVE_CARD_NO = 1;
if TEST_RESULT.SLAVE(SLAVE_CARD_NO) = ABSENT then
call DISPLAY (@NOT_PRESENT_7, GIM_AVAILABLE,
 TEXT, length(NOT_PRESENT_7));
else
do;
TMAP_RESULT.TMAP_SLAVE(SLAVE_CARD_NO) = UNTESTED;
TMAP_FLAG = INVOKE_SLAVE_TEST + SLAVE_CARD_NO;
call WAIT_FOR_TMAP;
if TMAP_RESULT.TMAP_SLAVE(SLAVE_CARD_NO) = PASSED then
call DISPLAY (@PASSED_7, GIM_AVAILABLE, TEXT, length(PASSED_7));
else
call DISPLAY (@FAILED_7, GIM_AVAILABLE, TEXT, length(FAILED_7));
end;
end; /* GIM 2 test */
do; /* F */
SLAVE_CARD_NO = 2;
if TEST_RESULT.SLAVE(SLAVE_CARD_NO) = ABSENT then
call DISPLAY (@NOT_PRESENT_8, GIM_AVAILABLE,
 TEXT, length(NOT_PRESENT_8));
else
do;
TMAP_RESULT.TMAP_SLAVE(SLAVE_CARD_NO) = UNTESTED;
TMAP_FLAG = INVOKE_SLAVE_TEST + SLAVE_CARD_NO;
call WAIT_FOR_TMAP;
if TMAP_RESULT.TMAP_SLAVE(SLAVE_CARD_NO) = PASSED then
call DISPLAY (@PASSED_8, GIM_AVAILABLE, TEXT, length(PASSED_8));
else
call DISPLAY (@FAILED_8, GIM_AVAILABLE, TEXT, length(FAILED_8));
end;
end; /* GIM 3 test */
do; /* G */
SLAVE_CARD_NO = 3;
if TEST_RESULT.SLAVE(SLAVE_CARD_NO) = AESENT then
    call DISPLAY (NOT_PRESENT_9, GIM_AVAILABLE,
                  TEXT, length(NOT_PRESENT_9));
else
    do;
        TMAP_RESULT.TMAP_SLAVE(SLAVE_CARD_NO) = UNTESTED;
        TMAP_FLAG = INVOKE_SLAVE_TEST + SLAVE_CARD_NO;
        call WAIT_FOR_TMAP;
        if TMAP_RESULT.TMAP_SLAVE(SLAVE_CARD_NO) = PASSED then
            call DISPLAY (PASSED_9, GIM_AVAILABLE, TEXT, length(PASSED_9));
        else
            call DISPLAY (FAILED_9, GIM_AVAILABLE, TEXT, length(FAILED_9));
    end;
end; /* SCMB test */

do; /* H */
   TMAP_RESULT.TMAP_API_BOARD = UNTESTED;
   TMAP_RESULT.TMAP_API_RAM = UNTESTED;
   TMAP_FLAG = INVOKE_API_TEST;
   call WAIT_FOR_TMAP;
   if TMAP_RESULT.TMAP_API_BOARD = PASSED then
       call DISPLAY (PASSED_10, GIM_AVAILABLE, TEXT, length(PASSED_10));
   else
       call DISPLAY (FAILED_10, GIM_AVAILABLE, TEXT, length(FAILED_10));
   end; /* MIDS test */

do; /* I */
   TMAP_RESULT.TMAP_SDB_BOARD = UNTESTED;
   TMAP_RESULT.TMAP_SDB_SELFTEST = UNTESTED;
   TMAP_RESULT.TMAP_SDB_RAM = UNTESTED;
   TMAP_FLAG = INVOKE_SDB_TEST;
   call WAIT_FOR_TMAP;
   if TMAP_RESULT.TMAP_SDB_BOARD = PASSED then
       call DISPLAY (PASSED_11, GIM_AVAILABLE, TEXT, length(PASSED_11));
   else
       call DISPLAY (FAILED_11, GIM_AVAILABLE, TEXT, length(FAILED_11));
   end; /* SDB test */
do; /* J */
TMAP_RESULT.TMAP_MSCC = UNTESTED;
TMAP_FLAG = INVOKE_MSCC_TEST;
call WAIT_FOR_TMAP;
if TMAP_RESULT.TMAP_MSCC = PASSED then
    call DISPLAY (@PASSED_12, GIM_AVAILABLE, TEXT, length(PASSED_12));
else
    call DISPLAY (@FAILED_12, GIM_AVAILABLE, TEXT, length(FAILED_12));
end; /* MSCC test */
end; /* case */
call GET_INPUT ('J');
do while (not LEGAL_INPUT ('A','J'));
    call GET_INPUT ('J');
end;
end; /* while LETTER <> ESC */
end TMAP_BIT;

$ eject
BIT: procedure;

declare BIT_MENU (*) byte data
(ESC, '1J', ESC, '1OH',
   'Primary processor off-line diagnostics menu', CR, LF,
   LF,
   'A Primary processor self-test', CR, LF,
   'B Primary processor EPROM/RAM test', CR, LF,
   'C Primary processor dynamic RAM test', CR, LF,
   'D First graphics interface module test', CR, LF,
   'E Second graphics interface module test', CR, LF,
   'F Third graphics interface module test', CR, LF,
   'G Serial communications to Multibus test', CR, LF,
   'H Applications processor interface test', CR, LF,
   'I System data bus controller test', CR, LF,
   'J Mass storage controller test', CR, LF,
   'K Secondary processor test', CR, LF,
   LF, EOM);

declare CONTINUE (*) byte data
(ESC, '1OH', 'Strike a key to continue...', EOM);

declare CPU_PASSED (*) byte data
(ESC, '1J', 'Main Applications Processor CPU has PASSED', CR, LF, EOM);

declare CPU_FAILED (*) byte data
(ESC, '1J', 'Main Applications Processor CPU has FAILED', CR, LF, EOM);

declare RAM_PASSED (*) byte data
(ESC, '1J', 'Global RAM has PASSED', CR, LF, EOM);

declare RAM_FAILED (*) byte data
(ESC, '1J', 'Global RAM has FAILED', CR, LF, EOM);

declare MSIC_PASSED (*) byte data
(ESC, '1J', 'MSIC has PASSED', CR, LF, EOM);

declare MSIC_FAILED (*) byte data
(ESC, '1J', 'MSIC has FAILED', CR, LF, EOM);

declare TMAP_CPU_FAILED (*) byte data
(ESC, '1J', 'TMAP_CPU has FAILED', CR, LF, EOM);

declare TMAP_CPU_PASSED (*) byte data
(ESC, '1J', 'TMAP_CPU has PASSED', CR, LF, EOM);

declare TMAP_RAM_FAILED (*) byte data
(ESC, '1J', 'TMAP_RAM has FAILED', CR, LF, EOM);

declare TMAP_RAM_PASSED (*) byte data
(ESC, '1J', 'TMAP_RAM has PASSED', CR, LF, EOM);

declare TMAP_RAM_PROMPT (*) byte data
(ESC, '1J', 'TMAP_RAM test invoked --- please be patient...', CR, LF, EOM);

declare TMAP_NOT_PRESENT (*) byte data
(ESC, '1J', 'TMAP not present in system', CR, LF, EOM);

declare TMAP_DRAM_FAILED (*) byte data
(ESC, '1J', 'TMAP_DRAM has FAILED', CR, LF, EOM);
declare TMAP_DRAM_PASSED (*) byte data
    (ESC, '[2J','TMAP_DRAM has PASSED', CR, LF, EOM);
declare DRAM_FAILED (*) byte data
    (ESC, '[2J','DRAM has FAILED', CR, LF, EOM);
declare DRAM_PASSED (*) byte data
    (ESC, '[2J','DRAM has PASSED', CR, LF, EOM);
declare MAP_DRAM_PROMPT (*) byte data
    (ESC, '[2J','MAP_DRAM test invoked --- please be patient...',
        CR, LF, EOM);
declare TMAP_DRAM_PROMPT (*) byte data
    (ESC, '[2J','TMAP_DRAM test invoked --- please be patient...',
        CR, LF, EOM);
declare TMAP_DRAM_MBUS_PASSED (*) byte data
    (ESC, '[2J','TMAP DRAM has PASSED on MULTIBUS', CR, LF, EOM);
declare DRAM_MBUS_PASSED (*) byte data
    (ESC, '[2J','DRAM has PASSED on MULTIBUS', CR, LF, EOM);
declare TMAP_DRAM_LBUS_PASSED (*) byte data
    (ESC, '[2J','TMAP DRAM has PASSED on LOCALBUS', CR, LF, EOM);
declare DRAM_LBUS_PASSED (*) byte data
    (ESC, '[2J','DRAM has PASSED on LOCALBUS', CR, LF, EOM);
declare SLAVE_CARD_NO byte;

TEST = TRUE;
TMAP_FLAG = SET_TEST_FLAG;
call ENABLE_INTERRUPTS;
call DISPLAY (@BIT_MENU, GIM_AVAILABLE, TEXT, length(BIT_MENU);
call DISPLAY (@SELECTION_PROMPT, GIM_AVAILABLE, TEXT, length(SELECTION_PROMPT));
call GET_INPUT ('K');
do while (not LEGAL_INPUT ('A','K'));
call GET_INPUT ('K');
end;
do while (LETTER <> ESC);
    BIT_INVOKED = TRUE; /* Set flag */
    do case SELECTION;
        do; /* A */
            TEST_RESULT.CPU_BOARD = UNTESTED;
            TEST_RESULT.CPU = UNTESTED;
            TEST_RESULT.ROM = UNTESTED;
            TEST_RESULT.MASTER_PIC = UNTESTED;
            TEST_RESULT.SLAVE_PIC = UNTESTED;
            TEST_RESULT.TMOUT_CIRCUIT = UNTESTED;
            TEST_RESULT.PIT = UNTESTED;
            TEST_RESULT.COPROCESSOR = UNTESTED;
            TEST_RESULT.MPSC = UNTESTED;
            TEST_RESULT.RAM = UNTESTED;
            TEST_RESULT.ACCESS_TO_1611 = UNTESTED;
            TEST_RESULT.LOCAL_BUS = UNTESTED;
            CLOCK_1_INTERRUPT = FALSE;
            CLOCK_2_INTERRUPT = FALSE;
            TMOUT_INTERRUPT = FALSE;
            output (STATUS_LATCH_ADDR) = RESET;
            TEST_RESULT.CPU = CPU_TEST;
            TEST_RESULT.RAM = RAM_TEST;
            call ON_BOARD_EEPROM_TEST;
            call MPIC_AND_TMOUT_TEST;
            call MPIC_AND_PIT_TEST;
            TEST_RESULT.MPSC = MPSC_IO_TEST (PORT_B);
            call SPIC_MPIC_AND_PIT_TEST;
            call COPROCESSOR_TEST;
            call TEST_ACCESS_TO_1611;
            call LOCAL_BUS_TEST;
            if TEST_RESULT.CPU and TEST_RESULT.ROM
                and TEST_RESULT.MASTER_PIC
                and TEST_RESULT.SLAVE_PIC
                and TEST_RESULT.TMOUT_CIRCUIT
                and TEST_RESULT.PIT
                and TEST_RESULT.COPROCESSOR
                and TEST_RESULT.MPSC
                and TEST_RESULT.RAM
                and TEST_RESULT.ACCESS_TO_1611
                and TEST_RESULT.LOCAL_BUS = PASSED then
                    TEST_RESULT.CPU_BOARD = PASSED;
                if TEST_RESULT.CPU_BOARD = PASSED then
                    call DISPLAY (@PASSED_3, GIM_AVAILABLE, TEXT, length(PASSED_3));
                else
                    call DISPLAY (@FAILED_3, GIM_AVAILABLE, TEXT, length(FAILED_3));
            end; /* MAP test */
do; /* B */
TEST_RESULT.ERAM = UNTESTED;
TEST_RESULT.EROM = UNTESTED;
TEST_RESULT.ERAM = ERAM_TEST;
call APPLICATIONS_ROM_TEST;
if TEST_RESULT.ERAM and TEST_RESULT.EROM = PASSED then
    call DISPLAY (@PASSED_4, GIM_AVAILABLE, TEXT, length(PASSED_4));
else
    call DISPLAY (@FAILED_4, GIM_AVAILABLE, TEXT, length(FAILED_4));
end; /* EPROM/RAM test */

do; /* C */
TEST_RESULT.DRAM_LBUS = UNTESTED;
TEST_RESULT.DRAM_MBUS = UNTESTED;
call POST_DRAM_TEST (LBUS);
call POST_DRAM_TEST (MBUS);
if TEST_RESULT.DRAM_LBUS and TEST_RESULT.DRAM_MBUS = PASSED then
    call DISPLAY (@PASSED_5, GIM_AVAILABLE, TEXT, length(PASSED_5));
else
    call DISPLAY (@FAILED_5, GIM_AVAILABLE, TEXT, length(FAILED_5));
end; /* DRAM test */

do; /* D */
SLAVE_CARD_NO = 0;
if TEST_RESULT.SLAVE(SLAVE_CARD_NO) = ABSENT then
    call DISPLAY (@NOT_PRESENT_6, GIM_AVAILABLE,
        TEXT, length(NOT_PRESENT_6));
else
    do;
        call TEST_SLAVE (SLAVE_CARD_NO);
        if TEST_RESULT.SLAVE(SLAVE_CARD_NO) = PASSED then
            call DISPLAY (@PASSED_6, GIM_AVAILABLE, TEXT, length(PASSED_6));
        else
            call DISPLAY (@FAILED_6, GIM_AVAILABLE, TEXT, length(FAILED_6));
        end;
    end; /* GIM 1 test */
do; /* E */
SLAVE_CARD_NO = 1;
if TEST_RESULT.SLAVE(SLAVE_CARD_NO) = ABSENT then
call DISPLAY (@NOT_PRESENT_7, GIM_AVAILABLE,
TEXT, length(NOT_PRESENT_7));
else
do;
call TEST_SLAVE(SLAVE_CARD_NO);
if TEST_RESULT.SLAVE(SLAVE_CARD_NO) = PASSED then
call DISPLAY (@PASSED_7, GIM_AVAILABLE, TEXT, length(PASSED_7));
else
call DISPLAY (@FAILED_7, GIM_AVAILABLE, TEXT, length(FAILED_7));
end;
end; /* GIM 2 test */
do; /* F */
SLAVE_CARD_NO = 2;
if TEST_RESULT.SLAVE(SLAVE_CARD_NO) = ABSENT then
call DISPLAY (@NOT_PRESENT_8, GIM_AVAILABLE,
TEXT, length(NOT_PRESENT_8));
else
do;
call TEST_SLAVE(SLAVE_CARD_NO);
if TEST_RESULT.SLAVE(SLAVE_CARD_NO) = PASSED then
call DISPLAY (@PASSED_8, GIM_AVAILABLE, TEXT, length(PASSED_8));
else
call DISPLAY (@FAILED_8, GIM_AVAILABLE, TEXT, length(FAILED_8));
end;
end; /* GIM 3 test */
do; /* G */
SLAVE_CARD_NO = 3;
if TEST_RESULT.SLAVE(SLAVE_CARD_NO) = ABSENT then
call DISPLAY (@NOT_PRESENT_9, GIM_AVAILABLE,
TEXT, length(NOT_PRESENT_9));
else
do;
call TEST_SLAVE(SLAVE_CARD_NO);
if TEST_RESULT.SLAVE(SLAVE_CARD_NO) = PASSED then
call DISPLAY (@PASSED_9, GIM_AVAILABLE, TEXT, length(PASSED_9));
else
call DISPLAY (@FAILED_9, GIM_AVAILABLE, TEXT, length(FAILED_9));
end;
end; /* SCMB test */
do; /* H */
TEST_RESULT.API_BOARD = UNTESTED;
TEST_RESULT.API_RAM = UNTESTED;
call TEST_API_RESPONSE;
TEST_RESULT.API_BOARD = TEST_RESULT.API_RAM;
call TIME (10000);
if TEST_RESULT.API_BOARD = PASSED then
    call DISPLAY (@PASSED_10, GIM_AVAILABLE, TEXT, length(PASSED_10));
else
    call DISPLAY (@FAILED_10, GIM_AVAILABLE, TEXT, length(FAILED_10));
end; /* MIDS test */
do;
/* I */
TEST_RESULT.SDB_BOARD = UNTESTED;
TEST_RESULT.SDB_SELFTEST = UNTESTED;
TEST_RESULT.SDB_RAM = UNTESTED;
call SDB_SELFTEST;
call SDB_RAM_TEST;
TEST_RESULT.SDB_BOARD = TEST_RESULT.SDB_SELFTEST
and TEST_RESULT.SDB_RAM;
if TEST_RESULT.SDB_BOARD = PASSED then
    call DISPLAY (@PASSED_11, GIM_AVAILABLE, TEXT, length(PASSED_11));
else
    call DISPLAY (@FAILED_11, GIM_AVAILABLE, TEXT, length(FAILED_11));
end; /* SDB test */
do; /* J */
TEST_RESULT.MSCC = UNTESTED;
output (MSCC_ADR) = TEST_MSCC;
call TIME (10000);
TEST_RESULT.MSCC = MSCC_STATUS_REGISTER;
if TEST_RESULT.MSCC = PASSED then
    call DISPLAY (@PASSED_12, GIM_AVAILABLE, TEXT, length(PASSED_12));
else
    call DISPLAY (@FAILED_12, GIM_AVAILABLE, TEXT, length(FAILED_12));
end; /* MSCC test */
do; /* K */
if TMAP_PRESENCE = PRESENT then
    do;
call TMAP_BIT;
call DISPLAY (@BIT_MENU, GIM_AVAILABLE, TEXT, length(BIT_MENU));
call DISPLAY (@SELECTION_PROMPT, GIM_AVAILABLE, TEXT, length(SELECTION_PROMPT));
end;
else
    call DISPLAY (@NOT_PRESENT_13, GIM_AVAILABLE, TEXT, length(NOT_PRESENT_13));
end; /* TMAP test */
end; /* case */
call ENABLE_INTERRUPTS;

/* Put the interrupt controller into a known state */

call GET_INPUT ('K');
do while (not LEGAL_INPUT ('A', 'K'));
   call GET_INPUT ('K');
end;

end; /* while LETTER <> ESC or LETTER <> 7 */

TEST = FALSE;
TMAP_FLAG = CLEAR_TEST_FLAG;

end BIT;

$ eject

SCMB_CPU_TEST: procedure;

SCMB_BUFFER (1).SCMB_CONTENTS (0) = UNTESTED;
SCMB_INTERRUPT = FALSE;
output (PIC_MASTER_8259A_ADR2) = UNMASK_TMOUT
   and UNMASK_CLOCK_1
   and UNMASK_MPSC
   and UNMASK_SLAVE;
output (PIC_SLAVE_8259A_ADR2) = UNMASK_SCMB;

CLOCK_INTERRUPT = FALSE;
do while CLOCK_INTERRUPT = FALSE;
   end; /* Start the clock */

TICK = 0.0;
TOCK = 0.0;
TEN_MILLISECONDS = 0;
SECONDS = 0;

output (SCMB_ADR) = PERFORM_CPU_TEST;
do while (SCMB_INTERRUPT <> TRUE) and (SECONDS < 2);
   end; /* Give the SCMB 2 seconds to report CPU test result */
output (PIC_MASTER_8259A_ADR2) = UNMASK_TMOUT
   and UNMASK_MPSC
   and UNMASK_SLAVE; /* Mask the clock */

if SCMB_INTERRUPT = TRUE then
   SCMB_INTERRUPT = FALSE;
else
   SCMB_BUFFER (1).SCMB_CONTENTS (0) = FAILED;

end SCMB_CPU_TEST;
SCMB_RAM_TEST: procedure;

declare I byte;
declare J byte;

SCMB_BUFFER (1).SCMB_CONTENTS (0) = UNTESTED;
SCMB_INTERRUPT = FALSE;
output (PIC_MASTER_8259A_ADR2) = UNMASK_TMOUT
and UNMASK_CLOCK_1
and UNMASK_MPSC
and UNMASK_SLAVE;
output (PIC_SLAVE_8259A_ADR2) = UNMASK_SCMB;

CLOCK_INTERRUPT = FALSE;
do while CLOCK_INTERRUPT = FALSE;
   end; /* Start the clock */
TICK = 0.0;
TOCK = 0.0;
TEN_MILLISECOSTS = 0;
SECONDS = 0;

output (SCMB_ADR) = PERFORM_RAM_TEST;
do while (SCMB_INTERRUPT <> TRUE) and (SECONDS < 6);
   end; /* Give the SCMB 6 seconds to report RAM test result */

output (PIC_MASTER_8259A_ADR2) = UNMASK_TMOUT
and UNMASK_MPSC
and UNMASK_SLAVE; /* Mask the clock */
if SCMB_INTERRUPT = TRUE then
   SCMB_INTERRUPT = FALSE;
else
   SCMB_BUFFER (1).SCMB_CONTENTS (0) = FAILED;
end SCMB_RAM_TEST;

$ eject
SET_LINK_SPEED: procedure;

declare SET_LINK_SPEED_HEADING (*) byte data
(ESC, '{2J', ESC, '{1;OH',
'Speed selection menu', CR, LF, LF,
'Current setting in Kb/s :', EOM);

declare SET_LINK_SPEED_MENU (*) byte data
(ESC, '{5;OH', CR, LF,
'A 15 B 20', CR, LF,
'C 25 D 30', CR, LF,
'E 35 F 40', CR, LF,
'G 45 H 50', CR, LF,
'I 55 J 60', CR, LF,
'K 65 L 70', CR, LF,
'M 75 N 80', CR, LF,
'O 90 P 100', CR, LF,
            CR, LF, EOM);

call DISPLAY (@SET_LINK_SPEED_HEADING, GIM_AVAILABLE,
            TEXT, length(SET_LINK_SPEED_HEADING));
call DISPLAY (@KBS_ARRAY, GIM_AVAILABLE, TEXT, length(KBS_ARRAY));
call DISPLAY (@SET_LINK_SPEED_MENU, GIM_AVAILABLE,
            TEXT, length(SET_LINK_SPEED_MENU));
call DISPLAY (@SELECTION_PROMPT, GIM_AVAILABLE,
            TEXT, length(SELECTION_PROMPT));

call GET_INPUT ('P');
do while SELECTION = INVALID_CHOICE;
call GET_INPUT ('P');
end;

if (SELECTION <> ESC) then
    do;
do case SELECTION;

    do; /* A */
        output (SCMB_ADR) = 41;
        CURRENT_SETTING = 15;
    end;

    do; /* B */
        output (SCMB_ADR) = 40;
        CURRENT_SETTING = 20;
    end;

end;
do;
output (SCMB_ADR) = 39;
CURRENT_SETTING = 25;
end;

/* C */
do;
output (SCMB_ADR) = 38;
CURRENT_SETTING = 30;
end;

/* D */
do;
output (SCMB_ADR) = 37;
CURRENT_SETTING = 35;
end;

/* E */
do;
output (SCMB_ADR) = 36;
CURRENT_SETTING = 40;
end;

/* F */
do;
output (SCMB_ADR) = 35;
CURRENT_SETTING = 45;
end;

/* G */
do;
output (SCMB_ADR) = 34;
CURRENT_SETTING = 50;
end;

/* H */
do;
output (SCMB_ADR) = 33;
CURRENT_SETTING = 55;
end;

/* I */
do;
output (SCMB_ADR) = 32;
CURRENT_SETTING = 60;
end;

/* J */
do;
output (SCMB_ADR) = 31;
CURRENT_SETTING = 65;
end;

/* K */
do;
output (SCMB_ADR) = 30;
CURRENT_SETTING = 70;
end;

/* L */
do; /* M */
output (SCMB_ADR) = 29;
CURRENT_SETTING = 75;
end;

do; /* N */
output (SCMB_ADR) = 28;
CURRENT_SETTING = 80;
end;

do; /* O */
output (SCMB_ADR) = 26;
CURRENT_SETTING = 90;
end;

do; /* P */
output (SCMB_ADR) = 24;
CURRENT_SETTING = 100;
end;

doi case */
end; /* if SELECTION <> ESC */
end SET_LINK_SPEED;

$ eject
HDLC_LINK_TEST: procedure;

   declare LINK_TEST_MENU (*) byte data
      (ESC, '[2J', ESC, '[1;1H', 'HDLC menu', CR, LF, LF, LF);

      ' Test  RX  TX  Calculated', CR, LF, LF, LF;
      ' No.  channel channel task speed Result', CR, LF, LF, LF;

      'A Test 1  1A -  2A', CR, LF, LF, LF;
      'B Test 2  1B -  2B', CR, LF, LF, LF;
      'C Test 3  2A -  1A', CR, LF, LF, LF;
      'D Test 4  2B -  1B', CR, LF, LF, LF;
      'E Select HDLC link speed', CR, LF, LF, LF;

      'Current HDLC link setting in Kb/s:', CR, LF, LF, LF;

   declare BAUD (*) byte data
      (' BAUD', EOM);

   declare LINK_TEST_PASSED (*) byte data
      (' PASSED', EOM);

   declare LINK_TEST_FAILED (*) byte data
      (' FAILED', EOM);

   declare POSITION_CURSOR_7 (*) byte data
      (ESC, '[7;25H', EOM);

   declare POSITION_CURSOR_8 (*) byte data
      (ESC, '[8;25H', EOM);

   declare POSITION_CURSOR_9 (*) byte data
      (ESC, '[9;25H', EOM);

   declare POSITION_CURSOR_10 (*) byte data
      (ESC, '[10;25H', EOM);

   declare SCMB_BUFFER_LENGTH real;
   declare REAL_TASK_SPEED real;
   declare TASK_SPEED word;
   declare TASK_ARRAY (7) byte;
   declare GPR_VALUE byte;
KBS_ARRAY (3) = EOM;
TASK_ARRAY (6) = EOM;
CURRENT_SETTING = 15;
call DISPLAY (@LINK_TEST_MENU, GIM_AVAILABLE, TEXT, length(LINK_TEST_MENU));
call C040_BIN_BYTE_TO_ASCII_DEC (CURRENT_SETTING, @KBS_ARRAY);
call DISPLAY (@KBS_ARRAY, GIM_AVAILABLE, TEXT, length(KBS_ARRAY));
call DISPLAY (@SELECTION_PROMPT, GIM_AVAILABLE, TEXT,
length(SELECTION_PROMPT));
call GET_INPUT ('E');
do while SELECTION = INVALID_CHOICE;
   call GET_INPUT ('E');
end;
do while LETTER <> ESC;
if ( (LETTER = 'E') or (LETTER = 4) ) then
do;
call SET_LINK_SPEED;
call DISPLAY (@LINK_TEST_MENU, GIM_AVAILABLE, TEXT, length(LINK_TEST_MENU));
call C040_BIN_BYTE_TO_ASCII_DEC (CURRENT_SETTING, @KBS_ARRAY);
call DISPLAY (@KBS_ARRAY, GIM_AVAILABLE, TEXT, length(KBS_ARRAY));
call DISPLAY (@SELECTION_PROMPT, GIM_AVAILABLE, TEXT,
length(SELECTION_PROMPT));
end;
else
do;
do case SELECTION ;
call DISPLAY (@POSITION_CURSOR_7, GIM_AVAILABLE,
TEXT, length(POSITION_CURSOR_7));
call DISPLAY (@POSITION_CURSOR_8, GIM_AVAILABLE,
TEXT, length(POSITION_CURSOR_8));
call DISPLAY (@POSITION_CURSOR_9, GIM_AVAILABLE,
TEXT, length(POSITION_CURSOR_9));
call DISPLAY (@POSITION_CURSOR_10, GIM_AVAILABLE,
TEXT, length(POSITION_CURSOR_10));
end; /* case */
output (PIC_MASTER_8259A_ADR2) = UNMASK_MPSM
and UNMASK_TMOUT
and UNMASK_CLOCK_1
and UNMASK_SLAVE ;
output (PIC_SLAVE_8259A_ADR2) = UNMASK_SMUB ;
CLOCK_INTERRUPT = FALSE; /* Start the clock */
do while CLOCK_INTERRUPT = FALSE;
   end;
TICK = 0.0;
TOCK = 0.0;
TEN_MILLISECONDS = 0;
SECONDS = 0;

GPR_VALUE = SELECTION + 20; /* Decide on test number to be performed */
SCMB_INTERRUPT = FALSE;
output (SCMB_ADR) = GPR_VALUE; /* Perform the test */

/* Now wait for SCMB to complete the test or for time limit to expire */
do while ((SCMB_INTERRUPT = FALSE) and (SECONDS < 10));
end;

output (PIC_MASTER_8259A_ADR2) = UNMASK_MPSC
and UNMASK_TMOUT
and UNMASK_SLAVE ;
output (PIC_SLAVE_8259A_ADR2) = UNMASK_SCMB ;

/* Mask off clock interrupt */
if SCMB_BUFFER (1).SCMB_CONTENTS (0) = PASSED then
do;
    SCMB_BUFFER_LENGTH = 16376.0; /* 2K byte buffer */
    if TICK = 0.0 then
        TASK_SPEED = 0; /* Avoid division by zero */
    else
        REAL_TASK_SPEED = (SCMB_BUFFER_LENGTH / TICK) * 100.0 ;
        TASK_SPEED = UNSIGN ( FIX ( REAL_TASK_SPEED ) );
call C030_BIN_WORD_TO_ASCII_DEC (TASK_SPEED, @TASK_ARRAY);
call DISPLAY (@TASK_ARRAY, GIM_AVAILABLE,
    TEXT, length(TASK_ARRAY));
call DISPLAY (@BAUD, GIM_AVAILABLE,
    TEXT, length(BAUD));
call DISPLAY (QLINK_TEST_PASSED, GIM_AVAILABLE,
    TEXT, length(LINK_TEST_PASSED));
else /* if SCMB_BUFFER(1).SCMB_CONTENTS (0) = PASSED */
    call DISPLAY (QLINK_TEST_FAILED, GIM_AVAILABLE,
        TEXT, length(LINK_TEST_FAILED));
end; /* else */
call GET_INPUT ('E');
do while SELECTION = INVALID_CHOICE;
    call GET_INPUT ('E');
end;
end; /* while LETTER <> ESC */

end HDLC_LINK_TEST;
SCMB_DEMONSTRATION: procedure;

declare SCMB_MENU (*) byte data
(ESC, '[2J', ESC, '[1;OH',
 'Serial communications to Multibus demonstration menu', CR, LF,
 CR, LF,
 'A Perform board self-test', CR, LF,
 'B Reset serial communications to Multibus card', CR, LF,
 'C Invoke HDLC menu', CR, LF,
 LF, EOM);

call DISPLAY (@SCMB_MENU, GIM_AVAILABLE, TEXT, length(SCMB_MENU));
call DISPLAY (@SELECTION_PROMPT, GIM_AVAILABLE, TEXT,
 length(SELECTION_PROMPT));
call GET_INPUT ('C');
do while SELECTION = INVALID_CHOICE;
call GET_INPUT ('C');
end;
do while LETTER <> ESC;
do case SELECTION ;
do; /* A */
call SCMB_CPU_TEST;
if SCMB_BUFFER (1).SCMB_CONTENTS (0) = PASSED then
do;
call SCMB_RAM_TEST;
if SCMB_BUFFER (1).SCMB_CONTENTS (0) = PASSED then
call DISPLAY (@PASSED_3, GIM_AVAILABLE,
 TEXT, length(PASSED_3));
else
call DISPLAY (@FAILED_3, GIM_AVAILABLE,
 TEXT, length(FAILED_3));
end;
else
call DISPLAY (@FAILED_3, GIM_AVAILABLE,
 TEXT, length(FAILED_3));
end;
do;
output (SCMB_ADR + 3) = TRUE; /* B */
call TIME (10000); /* Delay for 1 second while SCMB card resets */
output (SCMB_ADR) = REPORT_POST_RESULTS;
end;
do;
call HDLC_LINK_TEST; /* C */
call DISPLAY (@SCMB_MENU, GIM_AVAILABLE, TEXT, length(SCMB_MENU));
call DISPLAY (@SELECTION_PROMPT, GIM_AVAILABLE, TEXT,
    length(SELECTION_PROMPT));
end;

call GET_INPUT ('C');
do while SELECTION = INVALID_CHOICE;
call GET_INPUT ('C');
end;

call GET_INPUT ('C');
do while SELECTION = INVALID_CHOICE;
call GET_INPUT ('C');
end; /* while LETTER <> ESC */
end SCMB_DEMONSTRATION;

CONSOLE_DEMO: procedure ;

    output (STATUS_LATCH_ADR) = RESET_SYSTEM;

end CONSOLE_DEMO;
RBM_DEMO: procedure;

declare RBM_MENU (*) byte data
(ESC, '[2J', ESC, '[1;0H',
'Rollerball demonstration menu', CR, LF,
'Unmask rollerball', CR, LF,
'Mask rollerball', CR, LF,
'Retrieve rollerball module status', CR, LF,
LF, EOM);

declare RBM_CONNECTED (*) byte data
(ESC, '[5;30H', 'RBM connected', EOM);

declare RBM_NOT_CONNECTED (*) byte data
(ESC, '[5;30H', 'RBM not connected', EOM);

output (400h) = 38;

call WRITE_POLL (VDU, RBM_MENU);
call WRITE_POLL (VDU, SELECTION_PROMPT);

call GET_INPUT ('C');
do while SELECTION = INVALID_CHOICE;
  call GET_INPUT ('C');
end;

do while SELECTION <> ESC ;

do case SELECTION ;
  output (400h) = 40; /* A */
  output (400h) = 41; /* B */
  GRAPHICS_BUFFER (0).GPR_BUSY = TRUE;
  output (400h) = 39;
  do while GRAPHICS_BUFFER (0).GPR_BUSY = TRUE;
  end;
  if GRAPHICS_BUFFER (0).GRAPHICS_CONTENTS (25) = FALSE then
    call WRITE_POLL (VDU, RBM_NOT_CONNECTED);
  else
    call WRITE_POLL (VDU, RBM_CONNECTED);
  end;
end /* case */;
call GET_INPUT ('C');
do while SELECTION = INVALID_CHOICE;
  call GET_INPUT ('C');
end;
end RBM_DEMO;
MAIN: procedure public;

declare MAIN_MENU (*) byte data
(ESC, '[2J', ESC, '[1;OH',
 'CONSOLE DEMONSTRATION MAIN MENU', CR, LF,
 'A Perform rollerball demonstration', CR, LF,
 'B Display graphics test patterns', CR, LF,
 'C Demonstrate bus contention capabilities', CR, LF,
 'D Execute off-line built in tests', CR, LF,
 'E Test serial communications to Multibus', CR, LF,
 'F Return to Real Mode', CR, LF,
 ' (F resets the processor and allows the running of real mode code)', CR, LF,
 LF, EOM);

call DISPLAY (@MAIN_MENU, GIM_AVAILABLE, TEXT, length(MAIN_MENU));
call DISPLAY (@SELECTION_PROMPT, GIM_AVAILABLE:, TEXT,
 length(SELECTION_PROMPT));
call GET_INPUT ('F');
do while SELECTION = INVALID_CHOICE;
  call GET_INPUT ('F');
end;

if SELECTION <> ESC then

  do;
  do case SELECTION ;
    call RBM_DEMO; /* 0 */
    call TEST_GRAPHICS; /* 1 */
    call TEST_TMAP; /* 2 */
    call BIT; /* 3 */
    call SCMB_DEMONSTRATION; /* 4 */
    call CONSOLE_DEMO; /* 5 */
  end;

  end; /* if SELECTION <> ESC */

end MAIN;

discriminated

end PROTECTED_MODE_APPLICATIONS;
$ include (PLMPAR.INC)

/*******************************************************************************
 " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " 

*******************************************************************************/

* HISTORY  Version 1.0 :  
* 
*    Designed by : P.A. OLANDER    Date : August 1989  
*    Description : Original  
* 
*******************************************************************************
PROC2_CODE: do;

$ include (...)\STD\STDOEO.PDP$  
$ include (...)\STD\STDCONVT.PDP$  
$ include (...)\STD\STDBIT.PDP$  
$ include (...)\STD\STDPIC.PDP$  
$ include (...)\STD\STDCPU.PDP$  
$ include (...)\STD\STDRAM.PDP$  
$ include (...)\STD\STDERAM.PDP$  
$ include (...)\STD\PM_INIT.PDP$  
$ include (...)\STD\PM_SEGS.PDP$  

declare SECONDARY_PROC_EXECUTIVE label ;

declare IO_BASE_ADDRESS literally 'Od8h';
declare IO_INTERRUPT_NO literally '80';
declare IO_BAUD_RATE literally '9600';
declare INTERROGATION literally '51h';
declare HANDSHAKE literally '11';
declare TMAP_PRESENT literally '0ah';
declare NOT_BUSY literally '0';
declare TEXT literally '1';
declare KEYPRESS literally '2';
declare JOYSTICK literally '3';
declare SET literally '11';
declare EXIT literally '2';
declare MBUS literally '3';
declare GET literally '0';
declare PUT literally '1';
declare API_RAM_BUFFER_MIN literally '0h';
declare API_RAM_BUFFER_MAX literally '03feh';
declare SEGMENT_MIN literally '0h';
declare STATUS_LATCH_ADR literally '0e0h';
declare MSCC_ADR literally '600h';
declare MSCC_STATUS_REGISTER literally '601h';
declare TEST_MSCC literally '0fah';
declare ERASE_SCREEN (*) byte data
  (ESC, '2J', ESC, '0;OH', EOM);
declare POSITION_CURSOR (*) byte data
  (ESC, '24;60H', EOM);
declare DELIMITER (*) byte data (' ', EOM);
declare ASCII_VALUE (4) byte;
declare LETTER byte;
declare GPR_ADDRESS word;
declare FOREVER byte;
declare TMAP_MSG (*) byte data
  ('TMAP is running...', EOM);
declare TMOUT_INTERRUPT byte external;
declare CLOCK_1_INTERRUPT byte external;
declare CLOCK_2_INTERRUPT byte external;

INITIALISE_PORT_DATA: procedure;
  call INITIALISE_IO_ADDRESSES (PORT_A,
    IO_BASE_ADDRESS,
    IO_INTERRUPT_NO);
  call INITIALISE_PORT (PORT_A,
    IO_BAUD_RATE,
    EIGHT_DATA_BITS,
    TWO_STOP_BITS,
    NO_PARITY,
    TERMINAL);
  call INITIALISE_PORT (PORT_B,
    IO_BAUD_RATE,
    EIGHT_DATA_BITS,
    TWO_STOP_BITS,
    NO_PARITY,
    TERMINAL);
end INITIALISE_PORT_DATA;

declare DRAM_BUFFER_PTR pointer;
declare DRAM_BUFFER based DRAM_BUFFER_PTR (SEGMENT_MAX) byte;
declare DRAM_TEST_RESULT byte;
declare K integer;

$ eject
REPORT_POST_RESULTS: procedure public;

TMAP_FLAG = TMAP_PRESENT; /* Set presence indicator in global RAM */

TMAP_RESULT.TMAP_BOARD = BIT_RESULT.CPU_BOARD;
TMAP_RESULT.TMAP_CPU = BIT_RESULT.CPU;
TMAP_RESULT.TMAP_ROM = BIT_RESULT.ROM;
TMAP_RESULT.TMAP_MASTER_PIC = BIT_RESULT.MASTER_PIC;
TMAP_RESULT.TMAP_SLAVE_PIC = BIT_RESULT.SLAVE_PIC;
TMAP_RESULT.TMAP_TMOUT_CIRCUIT = BIT_RESULT.TMOUT_CIRCUIT;
TMAP_RESULT.TMAP_PIT = BIT_RESULT.PIT;
TMAP_RESULT.TMAP_COPROCESSOR = BIT_RESULT.COPROCESSOR;
TMAP_RESULT.TMAP_MPSC = BIT_RESULT.MPSC;
TMAP_RESULT.TMAP_RAM = BIT_RESULT.RAM;
TMAP_RESULT.TMAP_ACCESS_TO_1611 = BIT_RESULT.ACCESS_TO_1611;
TMAP_RESULT.TMAP_ERAM = BIT_RESULT.ERAM;
TMAP_RESULT.TMAP_EROM = BIT_RESULT.EROM;
TMAP_RESULT.TMAP_SDB_BOARD = BIT_RESULT.SDB_BOARD;
TMAP_RESULT.TMAP_LOCAL_BUS = BIT_RESULT.LOCAL_BUS;

end REPORT_POST_RESULTS;
TEST_API_RAM: procedure public;

 /*******************************************************************************
 * Tests the API dual port RAM from the secondary processor. *
 ********************************************************************************/

declare STORE byte;
declare I word;

TEST_RESULT.API_RAM = PASSED; /* Be optimistic */
I = API_RAM_BUFFER_MIN;
do while (I < API_RAM_BUFFER_MAX) and (TEST_RESULT.API_RAM <> FAILED)
   STORE = API_RAM_BUFFER(I);
   API_RAM_BUFFER(I) = ZERO_ONES;
   if (TMOUT_INTERRUPT = TRUE) or (API_RAM_BUFFER(I) <> ZERO_ONES) then 
do;
   TEST_RESULT.API_RAM = FAILED;
   TMOUT_INTERRUPT = FALSE;
   end;
else 
do;
   API_RAM_BUFFER(I) = ONE_ZEROS;
   if (TMOUT_INTERRUPT = TRUE) or (API_RAM_BUFFER(I) <> ONE_ZEROS) then 
do;
   TEST_RESULT.API_RAM = FAILED;
   TMOUT_INTERRUPT = FALSE;
   end;
else
   API_RAM_BUFFER(I) = STORE;
   I = I + 1;
end;
end; /* while */

end TEST_API_RAM;
/* Tests the intelligent slave boards from the secondary processor. */

declare SLAVE_CARD_NO byte;
declare SLAVE_ADDRESS word;
declare SLAVE_CPU byte;

if (SLAVE_CARD_NO >= 0) and (SLAVE_CARD_NO <= 3) then
do;
TEST_RESULT.SLAVE(SLAVE_CARD_NO) = UNTESTED;
SLAVE_RESULT.SLAVE_BOARD(SLAVE_CARD_NO) = UNTESTED;
do case SLAVE_CARD_NO;
    SLAVE_ADDRESS = 400h; /* GIM 1 */
    SLAVE_ADDRESS = 300h; /* GIM 2 */
    SLAVE_ADDRESS = 200h; /* GIM 3 */
    SLAVE_ADDRESS = 500h; /* SCMB */
end; /* case */
output (SLAVE_ADDRESS) = 7; /* CPU test : reports to DRAM */
call TIME (10000);
do while SLAVE_RESULT.SLAVE_BOARD(SLAVE_CARD_NO) = UNTESTED;
    call TIME (10000); /* Primary processor has higher priority, so delay */
end; /* Wait for slave CPU test result */
SLAVE_CPU = SLAVE_RESULT.SLAVE_BOARD(SLAVE_CARD_NO);

SLAVE_RESULT.SLAVE_BOARD(SLAVE_CARD_NO) = UNTESTED;
output (SLAVE_ADDRESS) = 8; /* RAM test : reports to DRAM */
call TIME (10000);
do while SLAVE_RESULT.SLAVE_BOARD(SLAVE_CARD_NO) = UNTESTED;
    call TIME (10000); /* Primary processor has higher priority, so delay */
end; /* Wait for slave CPU test result */

TEST_RESULT.SLAVE(SLAVE_CARD_NO) = SLAVE_CPU
    and SLAVE_RESULT.SLAVE_BOARD(SLAVE_CARD_NO);
TMAP_RESULT.TMAP_SLAVE(SLAVE_CARD_NO) = TEST_RESULT.SLAVE(SLAVE_CARD_NO);
end; /* if SLAVE_CARD_NO is valid */
end TEST_SLAVE;
PROC2_MAIN: procedure public;

/**************************************************************************
  * This is the main driving routine for the secondary processor. After
  * initialisation, it will remain in this procedure and service commands
  * from the host (primary) processor.
  *
**************************************************************************/

declare TITLE (*) byte data
(ESC, '4;OH','AIS DEMONSTRATION PACKAGE ', CR, LF,
'T_MAP card demonstration ', CR, LF, EOM);

declare MAP_MSG_1 (*) byte data
(ESC, '5;OH','Primary processor COUNT = XXXXX ',CR,EOM);

declare T_MAP_MSG_1 (*) byte data
(ESC, '8;OH','Secondary processor COUNT = ----- ',CR,EOM);

declare I byte;
declare ASCII_MAP_ARRAY (34) byte;
declare ASCII_TMAP_ARRAY (34) byte;
declare MAP_COUNTER word;
declare T_MAP_COUNTER word;
declare EXIT_TEST byte;
declare FLAG_STATUS byte;
declare SLAVE_CARD_NO byte;

call WRITE (VDU, @TITLE);
TMAP_FLAG = RESET;
MAP_COUNTER = 0;
T_MAP_COUNTER = 0;
do I = 0 to 33;
   ASCII_MAP_ARRAY (I) = MAP_MSG_1 (I);
   ASCII_TMAP_ARRAY (I) = T_MAP_MSG_1 (I);
end;

EXIT_TEST = FALSE;
do while EXIT_TEST = FALSE;
  FLAG_STATUS = TMAP_FLAG;
  if (FLAG_STATUS <> RESET) and (FLAG_STATUS < 5) then
    TMAP_FLAG = RESET;
  if (FLAG_STATUS > 15) then
    FLAG_STATUS = 0;
  TMOUT_INTERRUPT = FALSE;
do case FLAG_STATUS;
  
  ; /* Take no action */
end;

T_MAP_COUNTER = T_MAP_COUNTER + 1; /* Flag set */
do; /* Flag EXIT */
call WRITE (VDU, ERASE_SCREEN);
T_MAP_COUNTER = 0;
end; /* Flag EXIT */
do; /* Flag DISPLAY */
T_MAP_COUNTER = T_MAP_COUNTER + 1;
call C030_BIN_WORD_TO_ASCII_DEC (T_MAP_COUNTER, ASCII_TMAP_ARRAY(23));
call WRITE (VDU, ASCII_TMAP_ARRAY);
end; /* Flag DISPLAY */
T_MAP_COUNTER = 0; /* Case 4 */
do; /* Case 5: TMAP test */
TEST = TRUE;
TEST_RESULT.CPU_BOARD = UNTESTED;
TEST_RESULT.CPU = UNTESTED;
TEST_RESULT.ROM = UNTESTED;
TEST_RESULT.MASTER_PIC = UNTESTED;
TEST_RESULT.SLAVE_PIC = UNTESTED;
TEST_RESULT.TMOUT_CIRCUIT = UNTESTED;
TEST_RESULT.PIT = UNTESTED;
TEST_RESULT.COPROCESSOR = UNTESTED;
TEST_RESULT.MPSC = UNTESTED;
TEST_RESULT.RAM = UNTESTED;
TEST_RESULT.ACCESS_TO_1611 = UNTESTED;
TEST_RESULT.LOCAL_BUS = UNTESTED;
CLOCK_1_INTERRUPT = FALSE;
CLOCK_2_INTERRUPT = FALSE;
TMOUT_INTERRUPT = FALSE;
output (STATUS_LATCH_ADR) = RESET;
TEST_RESULT.CPU = CPU_TEST;
TEST_RESULT.RAM = MEM_TEST;
call ON_BOARDEPROM_TEST;
call MPIC_AND_TMOUT_TEST;
call MPIC_AND_PIT_TEST;
TEST_RESULT.MPSC = MPSC_IO_TEST (PORT_A);
TEST_RESULT.MPSC = MPSC_IO_TEST (PORT_B);
call SPIC_MPIC_AND_PIT_TEST;
call COPROCESSOR_TEST;
call TEST_ACCESS_TO_1611;
call LOCAL_BUS_TEST;
if TEST_RESULT.CPU
and TEST_RESULT.ROM
and TEST_RESULT.MASTER_PIC
and TEST_RESULT.SLAVE_PIC
and TEST_RESULT.TMOUT_CIRCUIT
and TEST_RESULT.PIT
and TEST_RESULT.COPROCESSOR
and TEST_RESULT.MPSC
and TEST_RESULT.RAM
and TEST_RESULT.ACCESS_TO_1611
and TEST_RESULT.LOCAL_BUS = PASSED then
TEST_RESULT.CPU_BOARD = PASSED;

TMAP_RESULT.TMAP_BOARD = TEST_RESULT.CPU_BOARD;
TMAP_RESULT.TMAP_CPU = TEST_RESULT.CPU;
TMAP_RESULT.TMAP_ROM = TEST_RESULT.ROM;
TMAP_RESULT.TMAP_MASTER_PIC = TEST_RESULT.MASTER_PIC;
TMAP_RESULT.TMAP_SLAVE_PIC = TEST_RESULT.SLAVE_PIC;
TMAP_RESULT.TMAP_TMOUT_CIRCUIT = TEST_RESULT.TMOUT_CIRCUIT;
TMAP_RESULT.TMAP_PIT = TEST_RESULT.PIT;
TMAP_RESULT.TMAP_COPROCESSOR = TEST_RESULT.COPROCESSOR;
TMAP_RESULT.TMAP_MPSC = TEST_RESULT.MPSC;
TMAP_RESULT.TMAP_RAM = TEST_RESULT.RAM;
TMAP_RESULT.TMAP_ACCESS_TO_1611 = TEST_RESULT.ACCESS_TO_1611;
TMAP_RESULT.TMAP_LOCAL_BUS = TEST_RESULT.LOCAL_BUS;
TMAP_FLAG = RESET;
end;

/* Case 6: EPROM/RAM test */
do;

TEST = TRUE;
TEST_RESULT.ERAM = UNTESTED;
TEST_RESULT.EROM = UNTESTED;
TEST_RESULT.ERAM = ERAM_TEST;
call APPLICATIONS_ROM_TEST;
TMAP_RESULT.TMAP_ERAM = TEST_RESULT.ERAM;
TMAP_RESULT.TMAP_EROM = TEST_RESULT.EROM;
TMAP_FLAG = RESET;
end;
do; /* Case 7: DRAM test */
TEST = TRUE;
TEST_RESULT.DRAM_LBUS = UNTESTED;
TEST_RESULT.DRAM_MBUS = UNTESTED;
call POST_DRAM_TEST (MBUS);
TMAP_RESULT.TMAP_DRAM_MBUS = TEST_RESULT.DRAM_MBUS;
TMAP_FLAG = RESET;
end;

do; /* Case 8: Gim 1 test */
TEST = TRUE;
SLAVE_CARD_NO = 0;
call TEST_SLAVE (SLAVE_CARD_NO);
TMAP_FLAG = RESET;
end;

do; /* Case 9: Gim 1 test */
TEST = TRUE;
SLAVE_CARD_NO = 1;
call TEST_SLAVE (SLAVE_CARD_NO);
TMAP_FLAG = RESET;
end;

do; /* Case 10: Gim 1 test */
TEST = TRUE;
SLAVE_CARD_NO = 2;
call TEST_SLAVE (SLAVE_CARD_NO);
TMAP_FLAG = RESET;
end;

do; /* Case 11: SCMB test */
TEST = TRUE;
SLAVE_CARD_NO = 3;
call TEST_SLAVE (SLAVE_CARD_NO);
TMAP_FLAG = RESET;
end;

do; /* Case 12: MIDS test */
TEST = TRUE;
TEST_RESULT.API_BOARD = UNTESTED;
TEST_RESULT.API_RAM = UNTESTED;
call TEST_API_RAM;
TEST_RESULT.API_BOARD = TEST_RESULT.API_RAM;
TMAP_RESULT.TMAP_API_BOARD = TEST_RESULT.API_BOARD;
TMAP_RESULT.TMAP_API_RAM = TEST_RESULT.API_RAM;
TMAP_FLAG = RESET;
end;
do; /* Case 13: SDB test */
TEST = TRUE;
TEST_RESULT.SDB_BOARD = UNTESTED;
TEST_RESULT.SDB_SELFTEST = UNTESTED;
TEST_RESULT.SDB_RAM = UNTESTED;
call SDB_SELFTEST;
call SDB_RAM_TEST;
TEST_RESULT.SDB_BOARD = TEST_RESULT.SDB_SELFTEST and TEST_RESULT.SDB_RAM;
TMAP_RESULT.TMAP_SDB_BOARD = TEST_RESULT.SDB_BOARD;
TMAP_RESULT.TMAP_SDB_SELFTEST = TEST_RESULT.SDB_SELFTEST;
TMAP_RESULT.TMAP_SDB_RAM = TEST_RESULT.SDB_RAM;
TMAP_FLAG = RESET;
end;

TEST = TRUE; /* Case 14 */
TEST = FALSE; /* Case 15 */

do; /* Case 16: MSCC test */
TEST_RESULT.MSCC = UNTESTED;
output (MSCC_ADR) = TEST_MSCC;
call TIME (10000);
TEST_RESULT.MSCC = MSCC_STATUS_REGISTER;
TMAP_RESULT.TMAP_MSCC = TEST_RESULT.MSCC;
TMAP_FLAG = RESET;
end; /* MSCC test */

end; /* case FLAG_STATUS */

end /* while EXIT_TEST = FALSE */;

end PROC2_MAIN;
disable;
call CLEAR$TASK$SWITCH$FLAG;

call INITIALISE_PORT_DATA;
output (PIC_MASTER_8259A_ADR2) = MASK_ALL_INTs;
output (PIC_SLAVE_8259A_ADR2) = MASK_ALL_INTs;
call UNMASK_PIC (TMOUT_INTERRUPT_NO);
call UNMASK_PIC (MPSC_INTERRUPT_NO);

enable;
TEST = TRUE;
/* This allows the secondary processor to handle a time-out
   if the primary processor times-out during POST */
do while TMAP_FLAG <> INTERROGATION;
   /* Wait for a command from the primary processor */
   end;
call REPORT_POST_RESULTS;

do while TMAP_FLAG <> HANDSHAKE;
   /* Wait for a handshake from the primary processor */
   end;

TEST = FALSE;
call WRITE_POLL (VDU, @CLEARSCREEN);
call WRITE_POLL (VDU, @TMAP_MSG);

FOREVER = 1;
do while FOREVER = 1;
   call PROC2_MAIN;
   end;

end PROC2_CODE;
2.3.2.2 ASM 286 files

2.3.2.2.1 PM_SEGS.ASM

$ debug
$ xref

; /***************************************************************************/
; * MODULE NAME : PROTECTED_MODE_SEGMENTS
; *
; /***************************************************************************/
; * Source Filename : PM_SEGS.ASM
; *
; * Source Compiler : ASM286
; *
; * Operating System : DOS 3.10
; *
; * Description : Defines segments for the protected mode code to enable absolute locating.
; *
; * Public procedures: None
; *
; * EPD files : None
; *
; * Include files : None
; *
; /***************************************************************************/

$ eject

; /***************************************************************************/
; * HISTORY Version 1.0 :
; *
; * Designed by : P.A. OLANDER Date : January 1990
; *
; * Description : Original
; *
; /***************************************************************************/
NAME  PROTECTED_MODE_SEGMENTS

TEST_RESULTS  struc
  CPU_BOARD     db ?
  CPU           db ?
  ROM           db ?
  MASTER_PIC    db ?
  SLAVE_PIC     db ?
  TMOUT_CIRCUIT db ?
  PIT           db ?
  COPROCESSOR   db ?
  MPSC          db ?
  RAM           db ?
  ACCESS_TO_1611 db ?
  ERAM          db ?
  EROM          db ?
  SDB_BOARD     db ?
  SDB_SELFTEST  db ?
  SDB_RAM       db ?
  LOCAL_BUS     db ?
  API_BOARD     db ?
  API_RAM       db ?
  API_EMAC      db ?
  EMAC_SWITCH   db 8 dup (?)
  STUCK_MODULE  db ?
  STUCK_KEY     db ?
  DRAM_MBUS     db ?
  DRAM_LBUS     db ?
  SLAVE         db 4 dup (?)
  MSCC          db ?
TEST_RESULTS  ends

TEST_RESULTS_SEG segment rw public
TEST_RESULT    TEST_RESULTS <>
public        TEST_RESULT
TEST_RESULTSSEG ends
TMAP_RESULTS struc
TMAP_BOARD db ?
TMAP_CPU db ?
TMAP_ROM db ?
TMAP_MASTER_PIC db ?
TMAP_SLAVE_PIC db ?
TMAP_TMOUT_CIRCUIT db ?
TMAP_PIT db ?
TMAP_COPROCESSOR db ?
TMAP_MPSX db ?
TMAP_RAM db ?
TMAP_ACCESS_TO_1611 db ?
TMAP_ERAM db ?
TMAP_EROM db ?
TMAP_SDB_BOARD db ?
TMAP_SDB_RAM db ?
TMAP_LOCAL_BUS db ?
TMAP_API_BOARD db ?
TMAP_API_RAM db ?
TMAP_DRAM_MBUS db ?
TMAP_DRAM_LBUS db ?
TMAP_SLAVE db 4 dup (?)
TMAP_MSCC db ?
TMAP_RESULTS ends

TMAP_GLOBAL_SEG segment rw
TMAP_FLAG db ?
TMAP_SCREEN db ?
TMAP_RESULT TMAP_RESULTS <>
public TMAP_FLAG
public TMAP_SCREEN
public TMAP_RESULT
TMAP_GLOBAL_SEG ends

GLOBAL_RAM segment rw
EXCHANGED_FLAG dw ?
SYNCHRONISATION_FLAG dw ?
STATUS_FLAG dw ?
public EXCHANGED_FLAG
public SYNCHRONISATION_FLAG
public STATUS_FLAG
GLOBAL_RAM ends
SLAVE_RESULTS struc
SLAVE_BOARD db 4 dup (?)
SLAVE_RESULTS ends

SLAVE_SEG segment rw
SLAVE_RESULT SLAVE_RESULTS <>
public SLAVE_RESULT
SLAVE_SEG ends

MAP_SLAVE_SEG segment rw
MAP_SLAVE_RESULT db 4 dup (?)
public MAP_SLAVE_RESULT
MAP_SLAVE_SEG ends

TMAP_SLAVE_SEG segment rw
TMAP_SLAVE_RESULT db 4 dup (?)
public TMAP_SLAVE_RESULT
TMAP_SLAVE_SEG ends

GRAPHICS_BUFFERS struc
BUFFER_FLAG db ?
DATA_TYPE db ?
GPR_BUSY db ?
GRAPHICS_CONTENTS db 2045 dup (?)
GRAPHICS_BUFFERS ends

SCMB_BUFFERS struc
SCMB_CONTENTS db 2048 dup (?)
SCMB_BUFFERS ends

SLAVE_BUFFER_SEG segment rw
GRAPHICS_BUFFER GRAPHICS_BUFFERS 3 dup (<>)
SCMB_BUFFER SCMB_BUFFERS 2 dup (<>)
public GRAPHICS_BUFFER
public SCMB_BUFFER
SLAVE_BUFFER_SEG ends

DRAM_MBUS_SEG segment rw
DRAM_MBUS_BUFFER_1 db 65535 dup (?)
public DRAM_MBUS_BUFFER_1
DRAM_MBUS_SEG ends

DRAM_LBUS_SEG segment rw
DRAM_LBUS_BUFFER_1 db 65535 dup (?)
public DRAM_LBUS_BUFFER_1
DRAM_LBUS_SEG ends
API_SEG_1 segment rw
API_RAM_BUFFER db 7fch dup (?)
public API_RAM_BUFFER
API_SEG_1 ends

API_SEG_2 segment rw
SKM1_CONTROL db 80h dup (?) ; BASE + 1000h - 107FH
SKM2_CONTROL db 80h dup (?) ; BASE + 1080h - 10FFH
SKM3_CONTROL db 80h dup (?) ; BASE + 1100h - 117FH
QKBM_CONTROL db 80h dup (?) ; BASE + 1180h - 11FFH
FILLER_1 db 80h dup (?) ; BASE + 1200h - 127FH
EMAC_CONTROL db 80h dup (?) ; BASE + 1280h - 12FFH
SPARE_CONTROL db 80h dup (?) ; BASE + 1300h - 137FH
SPARE_STATUS db 80h dup (?) ; BASE + 1380h - 13FFH
KEY_STATUS db 80h dup (?) ; BASE + 1400h - 147FH
QKBM_STATUS db 80h dup (?) ; BASE + 1480h - 14FFH
RBM_STATUS db 80h dup (?) ; BASE + 1500h - 157FH
EMAC_STATUS db 80h dup (?) ; BASE + 1580h - 15FFH
FILLER_2 db 100h dup (?) ; BASE + 1600h - 167FH
FILLER_3 db 0fch dup (?) ; BASE + 1700h - 177FH
API_WR_DATA dw ? ; BASE + 17FCh - 17FDH
AP_WR_DATA dw ? ; BASE + 17FEh - 17FFH
public SKM1_CONTROL
public SKM2_CONTROL
public SKM3_CONTROL
public QKBM_CONTROL
public EMAC_CONTROL
public SPARE_CONTROL
public SPARE_STATUS
public KEY_STATUS
public QKBM_STATUS
public RBM_STATUS
public EMAC_STATUS
public API_WR_DATA
public AP_WR_DATA
API_SEG_2 ends
end
2.3.2.3 EPD files

2.3.2.3.1 PM_APP.EPD

MAIN: procedure external;
    end MAIN;

2.3.2.3.2 PM_INIT.EPD

DISPLAY: procedure (SOURCE, SCREEN_NO, DATA_TYPE, DISTANCE) external;
    declare SOURCE pointer;
    declare MESSAGE based SOURCE byte;
    declare SCREEN_NO byte;
    declare DATA_TYPE byte;
    declare DISTANCE word;

    /* Put data into a buffer so that the graphics card can access it */

    end DISPLAY;

POST_DRAM_TEST: procedure (BUS_CHOICE) external;

/*********************************************************
* This procedure checks for stuck address lines by writing predefined *
* values to predefined locations. These locations have been specifically *
* chosen so as to exercise all the address lines. *
* **********************************************************/

    declare BUS_CHOICE byte;

    end POST_DRAM_TEST;

PM_INITIALISATION: procedure external;

    end PM_INITIALISATION ;
2.3.2.3.3 PM_SEGS.EPD

declare TEST_RESULT structure

(CPU_BOARD byte,
 CPU byte,
 ROM byte,
 MASTER_PIC byte,
 SLAVE_PIC byte,
 TMOUT_CIRCUIT byte,
 PIT byte,
 COPROCESSOR byte,
 MPSC byte,
 RAM byte,
 ACCESS_TO_1611 byte,
 ERAM byte,
 EROM byte,
 SDB_BOARD byte,
 SDB_SELFTEST byte,
 SDB_RAM byte,
 LOCAL_BUS byte,
 API_BOARD byte,
 API_RAM byte,
 API_EMAC byte,
 EMAC_SWITCH (8) byte,
 STUCK_MODULE byte,
 STUCK_KEY byte,
 DRAM_MBUS byte,
 DRAM_LBUS byte,
 SLAVE (4) byte,
 MSCC byte ) external;
declare TMAP_RESULT structure

(TMAP_BOARD  byte,
 TMAP_CPU    byte,
 TMAP_ROM    byte,
 TMAP_MASTER_PIC byte,
 TMAP_SLAVE_PIC byte,
 TMAP_TMOUT_CIRCUIT byte,
 TMAP_PIT    byte,
 TMAP_COPROCESSOR byte,
 TMAP_MPSC   byte,
 TMAP_RAM    byte,
 TMAP_ACCESS_TO_1611 byte,
 TMAP_ERAM   byte,
 TMAP_EROM   byte,
 TMAP_SDB_BOARD byte,
 TMAP_SDB_SELFTEST byte,
 TMAP_SDB_RAM byte,
 TMAP_LOCAL_BUS byte,
 TMAP_API_BOARD byte,
 TMAP_API_RAM byte,
 TMAP_DRAM_MBUS byte,
 TMAP_DRAM_LBUS byte,
 TMAP_SLAVE  (4) byte,
 TMAP_MSCC   byte ) external;

declare TMAP_FLAG byte external;
declare TMAP_SCREEN byte external;

declare SLAVE_RESULT structure

(SLAVE_BOARD  (4) byte ) external;

declare MAP_SLAVE_RESULT (4) byte external;
declare TMAP_SLAVE_RESULT (4) byte external;

declare GRAPHICS_BUFFER (3) structure

(BUFFER_FLAG  byte,
 DATA_TYPE    byte,
 GPR_BUSY     byte,
 GRAPHICS_CONTENTS (2045) byte ) external;

declare SCMB_BUFFER (2) structure

(SCMB_CONTENTS (2048) byte ) external;

declare DRAM_MBUS_BUFFER_1 (0xffff) byte external;
declare DRAM_LBUS_BUFFER_1 (0xffff) byte external;
declare API_RAM_BUFFER (07fch) byte external;

declare SKM1_CONTROL (80h) byte; /* BASE + 1000h - 107FH */
declare SKM2_CONTROL (80h) byte; /* BASE + 1080h - 10FFH */
declare SKM3_CONTROL (80h) byte; /* BASE + 1100h - 117FH */
declare QKBM_CONTROL (80h) byte; /* BASE + 1180h - 11FFH */
declare FILLER_1 (80h) byte; /* BASE + 1200h - 127FH */
declare EMAC_CONTROL (80h) byte; /* BASE + 1280h - 12FFH */
declare SPARE_CONTROL (80h) byte; /* BASE + 1300h - 137FH */
declare SPARE_STATUS (80h) byte; /* BASE + 1380h - 13FFH */
declare KEY_STATUS (80h) byte; /* BASE + 1400h - 147FH */
declare QKBM_STATUS (80h) byte; /* BASE + 1480h - 14FFH */
declare RBM_STATUS (80h) byte; /* BASE + 1500h - 157FH */
declare EMAC_STATUS (80h) byte; /* BASE + 1580h - 15FFH */
declare API_WR_DATA word; /* BASE + 17FCh - 17FDH */
declare AP_WR_DATA word; /* BASE + 17FEh - 17FFH */
APPENDIX D : CREATION OF THE EPROMS

1 INTRODUCTION ........................................................... 2

2 CREATION OF THE EPROMS ON THE PROCESSOR CARDS ............... 3
   2.1 STANDARDISED CODE LINKABLE FILE CREATION .................. 3
   2.2 STANDARDISED CODE LOCATABLE FILE CREATION ............... 3
   2.3 STANDARDISED CODE BINARY FILE CREATION .................... 5
   2.4 EPROM LOADING PROCEDURE FOR THE STANDARDISED CODE ...... 5

3 CREATION OF THE EPROMS ON THE EPROM/RAM CARDS ................. 7
   3.1 LINKING THE REAL MODE SUBSYSTEM-SPECIFIC CODE ............ 7
   3.2 LOCATING THE REAL MODE SUBSYSTEM-SPECIFIC CODE .......... 8
   3.3 REAL MODE SUBSYSTEM-SPECIFIC BINARY FILE CREATION ....... 9
   3.4 BINDING THE PRIMARY PROCESSOR PROTECTED MODE CODE ...... 10
   3.5 BUILDING THE PRIMARY PROCESSOR PROTECTED MODE CODE ... 10
   3.6 BINDING THE SECONDARY PROCESSOR PROTECTED MODE CODE .... 14
   3.7 BUILDING THE SECONDARY PROCESSOR PROTECTED MODE CODE .. 15
   3.8 PREPARING THE PROTECTED MODE CODE FOR THE EPROMS ..... 18
   3.9 EPROM LOADING PROCEDURE FOR THE SUBSYSTEM-SPECIFIC CODE . 18

4 REFERENCES ........................................................................ 20
In order to insert the designed code into EPROMs, the procedure illustrated in Figure 1 was followed for the standardised code and the real mode subsystem-specific code. The procedure illustrated in Figure 2 was followed to prepare the protected mode code associated with both processor cards for the EPROM/RAM cards. Each of the steps shown in these illustrations are described in the following sections for the creation of the standardised code, the real mode subsystem-specific code and the protected mode code applicable to each processor card, together with a presentation of the EPROM loading procedures.

**Figure 1**: EPROM loadable module creation procedure for 8086 object modules

**Figure 2**: EPROM loadable module creation procedure for 80286 object modules
2 CREATION OF THE EPROMS ON THE PROCESSOR CARDS

2.1 STANDARDISED CODE LINKABLE FILE CREATION

Following compilation or assembly, the object modules were combined via the Intel 8086 Linker utility. STDLNK.BAT was a batch file that invoked the following commands:

```
link86 STDEEXEC.OBJ, &
STDMIN.OBJ, &
STDCPU.OBJ, &
STDRA.M.OBJ, &
STDERAM.OBJ, &
STDBIT.OBJ, &
STDPC1C.OBJ, &
STDCIT.OBJ, &
STDCPTAT.OBJ, &
STDDINTS.OBJ, &
STDIAGS.OBJ, &
STDCONV.T.OBJ, &
PLM86.LIB, &
8087.LIB, &
DCON87.LIB, &
CEL87.LIB, &
EHE87.LIB, &
to STDCODE.LNK
```

exit

2.2 STANDARDISED CODE LOCATABLE FILE CREATION

The standardised link file was located using the Intel8086 Locater by invoking the STDLOC.BAT batch file. Initially, this batch file was required to produce code that would load successfully onto the RAM of the CPU card. The Intel in-circuit emulator (ICE) was used to debug the code.

Following several iterations in the debugging cycle, the locater batch file was finally edited to produce code that would load onto the ROM of the CPU cards. The final revision of this batch file generated the following commands:

```
LOC86 STDCODE.LNK to STDCODE.LOC &
addresses( segments

(RAM_BLOCK  ( 0h ), &
EXEC_DATA  ( 2000h ), &
DATA  ( 2100h ), &
BIT_DATA  ( 2200h ), &
```
D-4

INIT_DATA (2300h), &
PIC_DATA (2400h), &
PIT_DATA (2500h), &
STATUS_LATCH_DATA (2600h), &
INTS_DATA (2700h), &
IO_8274_DATA (2800h), &
CONV_DATA (2900h), &
BIT_RESULTS_SEG (3000h), &
NO_OF_RUNS_SEG (3100h), &
DIAG_DATA (3200h), &
STACK (3300h), &
SDB_BUFFER (3400h), &
EROM_BLOCK_1 (3500h), &
EROM_BLOCK_2 (3600h), &
EROM_BLOCK_3 (3700h), &
EROM_BLOCK_4 (3800h), &
EROM_BLOCK_5 (3900h), &
EROM_BLOCK_6 (3A00h), &
EROM_BLOCK_7 (3B00h), &
EROM_BLOCK_8 (3C00h), &
SYSTEM_DESC (3D00h), &
EROM_BLOCK_9 (3E00h), &
EROM_BLOCK_10 (3F00h), &
DIAG_CODE (4000h), &
LIB_87_INIT (4100h), &
LIB_87_PUB (4200h), &
PIT_CODE (4300h), &
PIC_CODE (4400h), &
STATUS_LATCH_CODE (4500h), &
CONV_CODE (4600h), &
CODE (4700h), &
INIT_CODE (4800h), &
BIT_CODE (4900h), &
INTS_CODE (4A00h), &
IO_8274_CODE (4B00h), &
EXEC_CODE (4C00h)

start (EXECUTIVE) &
initcode (F5000h) &
bootstrap
exit
2.3 STANDARDISED CODE BINARY FILE CREATION

Preparing the standardised code for the EPROMS required that the object module format file output by the locator be converted to hexadecimal and then finally to binary. Both of these steps were fairly elementary, viz.:

(a) the code was converted to a hexadecimal file by using the Intel object to hexadecimal file converter and invoking the command "OI-I86 STDCODE.LOC". This resulted in the output file STDCODE.HEX.

(b) the binary file (STDCODE.BIN) to be loaded into the EPROMs was created using the Intel hexadecimal to binary converter and invoking the command "HEXBIN2 STDCODE.HEX".

2.4 EPROM LOADING PROCEDURE FOR THE STANDARDISED CODE

The "HI-LO" EPROM programmer unit [INTEL, 1987] was connected via the connecting cable and the programmer system adaptor card was plugged into the personal computer.

After invocation of the Intel EPROM programming software, the following procedure was followed:

1) The programmer was initialised to the correct manufacturer and chip.

2) The binary file STDCODE.BIN was loaded into the buffer of the personal computer at buffer address 0h.
3) The memory buffer was displayed and values of FFh were inserted at the checksum addresses. Since the POST calculated the checksum and informed the user via the RS232 regarding the calculated value, it was possible to update these values at a later stage. The appropriate addresses are shown in Table I.

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFFDh</td>
<td>Low checksum byte</td>
</tr>
<tr>
<td>FFFEh</td>
<td>High checksum byte</td>
</tr>
<tr>
<td>FFFFFh</td>
<td>1 = Checksums are present</td>
</tr>
</tbody>
</table>

4) The EPROMs were each inserted into the "HI-LO" EPROM programmer unit as shown in Figure 3 and programmed.

5) Following insertion of the fresh EPROMs into the appropriate slots on the CPU card, the system description area normally resident on the EPROMs of the EPROM/RAM card were mapped to the emulator high speed memory. Appropriate values were chosen and dummy code was loaded into the RAM of the EPROM/RAM card. This enabled full debugging of the ROM based code and the determination of the desired checksum information, which was then reprogrammed into the EPROMs.
3 CREATION OF THE EPROMS ON THE EPROM/RAM CARDS

3.1 LINKING THE REAL MODE SUBSYSTEM-SPECIFIC CODE

After compilation or assembly, the real mode subsystem-specific object modules were combined using the Intel 8086 Linker utility. LNK_RM.BAT was a batch file that invoked the following commands:

```
LINK86 RM_EXEC.OBJ, &
  RM_APP.OBJ, &
  RM_BIT.OBJ, &
  RM_COPY.OBJ, &
  RM_MSCC.OBJ, &
  RM_API.OBJ, &
  RM_INTS.OBJ, &
  RM_SEGS.OBJ, &
  ..\STD\STDINIT.OBJ, &
  ..\STD\STDCPU.OBJ, &
  ..\STD\STDRAM.OBJ, &
  ..\STD\STDERAM.OBJ, &
  ..\STD\STDSDB.OBJ, &
  ..\STD\STDBIT.OBJ, &
  ..\STD\STDPIC.OBJ, &
  ..\STD\STDPIT.OBJ, &
  ..\STD\STDSTAT.OBJ, &
  ..\STD\STDINTS.OBJ, &
  ..\STD\STDIO.OBJ, &
  ..\STD\STDCONVT.OBJ, &
 D:\SWB6\PLM86.LIB, &
 D:\SWB6\8087.LIB, &
 D:\SWB6\DCON87.LIB, &
 D:\SWB6\CEL87.LIB, &
 D:\SWB6\EHI87.LIB &
TO RM_CODE
EXIT
CTTY CON
```
3.2 LOCATING THE REAL MODE SUBSYSTEM-SPECIFIC CODE

The real mode subsystem-specific link file was located using the Intel 8086 Locater by invoking the LOC_RM.BAT batch file. Initially, this batch file was required to produce code that would load successfully onto the RAM of the EPROM/RAM card. The Intel in-circuit emulator was also used to debug this code.

After several iterations in the debugging cycle, the locater batch file was finally edited to produce code that would load onto the ROM of the EPROM/RAM cards. The final revision of this batch file generated the following commands:

```
LOC86 RM_CODE TO RM_CODE.LOC

ADDITIONS(SEGMENTS)
REAL_MODE_EXECUTIVE_CODE (C0100h), &
REAL_MODE_APPLICATIONS_CODE (C1000h), &
MSCC_APPLICATION_CODE (C2000h), &
REAL_MODE_BIT_CODE (C3000h), &
REAL_MODE_INTERRUPTS_CODE (C4000h), &
COPY_CODE (C5000h), &
API_APPLICATIONS_CODE (C6000h), &
REAL_MODE_EXECUTIVE_DATA (40080h), &
REAL_MODE_APPLICATIONS_DATA (40100h), &
REAL_MODE_BIT_DATA (40200h), &
REAL_MODE_INTERRUPTS_DATA (40300h), &
API_APPLICATIONS_DATA (40400h), &
MSCC_APPLICATION_DATA (40800h), &
BIT_CODE (F8000h), &
INIT_CODE (F9000h), &
CODE (F8000h), &
PIC_CODE (F6300h), &
PIT_CODE (F6000h), &
STATUS_LATCH_CODE (F6600h), &
INTS_CODE (F0000h), &
LIB_87_PUB (F4000h), &
LIB_87_INIT (F3000h), &
IO_8274_CODE (FE000h), &
CONV_CODE (F7000h), &
STACK (D0000h), &
DATA (2100h), &
BIT_DATA (2200h), &
INIT_DATA (2300h), &
PIC_DATA (2400h), &
PIT_DATA (2500h), &
STATUS_LATCH_DATA (2600h), &
INTS_DATA (2700h), &
IO_8274_DATA (3000h), &
CONV_DATA (4000h), &
```
Preparing the subsystem-specific real mode code for the EPROMS required that the object module format file output by the locator be converted to hexadecimal and then finally to binary. Again, both of these steps were fairly elementary, viz.:

(a) the code was converted to a hexadecimal file by using the Intel object to hexadecimal file converter and invoking the command "OH86 RM_CODE.LOC". This resulted in the output file RM_CODE.HEX.

(b) the binary file (RM_CODE.BIN) to be loaded into the EPROMs was created using the Intel hexadecimal to binary converter and invoking the command "HEXBIN2 RM_CODE.HEX".
3.4 BINDING THE PRIMARY PROCESSOR PROTECTED MODE CODE

The standardised code was re-compiled and re-assembled using the PLM286 and ASM286 commands respectively, thus generating 80286 object module files. These files were then bound to the 80286 object module files associated with the primary processor by executing the BNDP1.BAT batch file. This batch file generated the following command file:

```
BND286 PM_EXEC.OBJ, &
    PM_INIT.OBJ, &
    PM_INTS.OBJ, &
    PM_APP.OBJ, &
    PM_SEGS.OBJ, &
    STDCPU.OBJ, &
    STORAM.OBJ, &
    STDERAM.OBJ, &
    STDBIT.OBJ, &
    STDPIC.OBJ, &
    STDINTS.OBJ, &
    STDSTAT.OBJ, &
    STDIO.OBJ, &
    STDCONVMT.OBJ &
    NOLOAD &
    NAME(PM_CODE) &
    OBJECT(PM_CODE) &
EXIT
CTTY CON
```

3.5 BUILDING THE PRIMARY PROCESSOR PROTECTED MODE CODE

The output of the 80286 binder was bound to the 80286 object module file that changed the 80286 processor from real mode to 80286 protected mode by filtering the two modules through a building procedure. This build file also catered for the absolute locating of appropriate segments as well as defining attributes related to protected mode processing. The command and build files are shown below:

```
BLD286 PM_ENTP.OBJ, &
    PM_CODE &
    BF(PROC1.BLD) &
    OBJECT(PM_PROC1.LOC)
EXIT
CTTY CON
```
PROC1_BLD;

SEGMENT

*SEGMENTS (DPL = 0),

ENTER_PROTECTED_MODE_CODE (DPL = 0, BASE = 0AFF00H),

ENTER_PROTECTED_MODE_STACK (DPL = 0, PRESENT,
BASE = 0EO00H,
LIMIT = 1FFFH, ED),

PM_CODE_STACK (DPL = 0, PRESENT,
BASE = 0EO00H,
LIMIT = 1FFFH, ED),

BIT_RESULTS_SEG (DPL = 0, BASE = 5000H),
TEST_RESULTS_SEG (DPL = 0, BASE = 5000H),
PROC_ID_SEG (DPL = 0, BASE = 5200H),
SLAVE_SEG (DPL = 0, BASE = 5280H),

GLOBAL_RAM (DPL = 0, BASE = 11000H),
PROC2_GLOBAL_SEG (DPL = 0, BASE = 11220H),
API_SEG_1 (DPL = 0, BASE = 11000H),
API_SEG_2 (DPL = 0, BASE = 20800H),
SUB_BUFFER (DPL = 0, BASE = 11000H),

MAP_SLAVE_SEG (DPL = 0, BASE = 40000H),

RAM_BLOCK (DPL = 0, BASE = 00H, LIMIT = OFFFFH),
ERAM_BLOCK_1 (DPL = 0, BASE = 40000H, LIMIT = OFFFFH),
ERAM_BLOCK_2 (DPL = 0, BASE = 50000H, LIMIT = OFFFFH),
ERAM_BLOCK_3 (DPL = 0, BASE = 60000H, LIMIT = OFFFFH),
ERAM_BLOCK_4 (DPL = 0, BASE = 70000H, LIMIT = OFFFFH),
ERAM_BLOCK_5 (DPL = 0, BASE = 80000H, LIMIT = OFFFFH),
ERAM_BLOCK_6 (DPL = 0, BASE = 90000H, LIMIT = OFFFFH),

EROM_BLOCK_1 (DPL = 0, BASE = 60000H, LIMIT = OFFFFH),
EROM_BLOCK_2 (DPL = 0, BASE = 70000H, LIMIT = OFFFFH),
EROM_BLOCK_3 (DPL = 0, BASE = 80000H, LIMIT = OFFFFH),
EROM_BLOCK_4 (DPL = 0, BASE = 90000H, LIMIT = OFFFFH),
EROM_BLOCK_5 (DPL = 0, BASE = A0000H, LIMIT = OFFFFH),
EROM_BLOCK_6 (DPL = 0, BASE = B0000H, LIMIT = OFFFFH),
EROM_BLOCK_7 (DPL = 0, BASE = C0000H, LIMIT = OFFFFH),
EROM_BLOCK_8 (DPL = 0, BASE = D0000H, LIMIT = OFFFFH),
SYSTEM_DESC (DPL = 0, BASE = DFFFFH, LIMIT = 00H),

EROM_BLOCK_9 (DPL = 0, BASE = E0000H, LIMIT = OFFFFH),
EROM_BLOCK_10 (DPL = 0, BASE = F0000H, LIMIT = OFFFFH),
DRAM_LBUS_SEG (DPL = 0, BASE = 100000H),
DRAM_MBUS_SEG (DPL = 0, BASE = 300000H),
PROC2_SLAVE_SEG (DPL = 0, BASE = 360000H),
SLAVE_BUFFER_SEG (DPL = 0, BASE = 350000H);

TASK

TASK_1 (OBJECT = ENTER_PROTECTED_MODE,
          DPL = 0,
          BASE = 1C00H,
          LIMIT = 100,
          NOT INTENABLE );

SYSTEM_EXECUTIVE (OBJECT = PM_CODE,
                   DPL = 0,
                   BASE = 1D00H,
                   LIMIT = 100,
                   STACKS = (PM_CODE.STACK),
                   NOT INTENABLE );

GATE

SYSTEM_EXECUTIVE (TASK, ENTRY = SYSTEM_EXECUTIVE);

TABLE

LDT_2 ( DPL = 0, PRESENT, BASE = OCOOH, LIMIT = 100,
        ENTRY = (99:SYSTEM_EXECUTIVE) );


<table>
<thead>
<tr>
<th>GATE</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DIVIDE_ERROR(INTERRUPT),</td>
<td>SINGLE_STEP(INTERRUPT),</td>
</tr>
<tr>
<td>NMI(INTERRUPT),</td>
<td>BREAKPOINT(INTERRUPT),</td>
</tr>
<tr>
<td>INTO_OVERFLOW(INTERRUPT),</td>
<td>BOUND_RANGE(INTERRUPT),</td>
</tr>
<tr>
<td>INVALID_OPCODE(INTERRUPT),</td>
<td>PROC_EXT_NOT_AVAILABLE(INTERRUPT),</td>
</tr>
<tr>
<td>DOUBLE_EXCEPTION(INTERRUPT),</td>
<td>PROC_EXT_SEGMENT_OVERRUN(INTERRUPT),</td>
</tr>
<tr>
<td>INVALID_TASK(INTERRUPT),</td>
<td>SEGMENT_NOT_PRESENT(INTERRUPT),</td>
</tr>
<tr>
<td>STACK_OVERRUN(INTERRUPT),</td>
<td>GENERAL_PROTECTION(INTERRUPT),</td>
</tr>
<tr>
<td>NCP_INTERRUPT(INTERRUPT),</td>
<td>CHA_TX_EMPTY(INTERRUPT),</td>
</tr>
<tr>
<td>CHB_TX_EMPTY(INTERRUPT),</td>
<td>CHA_RX_AVAIL(INTERRUPT),</td>
</tr>
<tr>
<td>CHB_RX_AVAIL(INTERRUPT),</td>
<td>CHA_RX_ERROR(INTERRUPT),</td>
</tr>
<tr>
<td>CHB_RX_ERROR(INTERRUPT),</td>
<td></td>
</tr>
<tr>
<td>TMOUT(INTERRUPT),</td>
<td>REAL_TIME_CLOCK (INTERRUPT),</td>
</tr>
<tr>
<td>MASTER_INTERRUPT_2(INTERRUPT),</td>
<td>MASTER_INTERRUPT_3(INTERRUPT),</td>
</tr>
<tr>
<td>MASTER_INTERRUPT_4(INTERRUPT),</td>
<td>MASTER_INTERRUPT_7(INTERRUPT),</td>
</tr>
<tr>
<td>ILLEGAL_INT(INTERRUPT),</td>
<td></td>
</tr>
<tr>
<td>API(INTERRUPT),</td>
<td>SCMB(INTERRUPT),</td>
</tr>
<tr>
<td>SLAVE_INTERRUPT_2(INTERRUPT),</td>
<td>PROC2_INTERRUPT(INTERRUPT),</td>
</tr>
<tr>
<td>SLAVE_CLOCK(INTERRUPT),</td>
<td>SLAVE_INTERRUPT_5(INTERRUPT),</td>
</tr>
<tr>
<td>SLAVE_INTERRUPT_6(INTERRUPT),</td>
<td>SLAVE_INTERRUPT_7(INTERRUPT)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TABLE</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>GDT ( DPL = 0, PRESENT, BASE = 800H, LIMIT = 100, ENTRY = (LDT_2)),</td>
<td></td>
</tr>
<tr>
<td>IDT ( DPL = 0, PRESENT, BASE = 0H, LIMIT = 256,</td>
<td></td>
</tr>
<tr>
<td>ENTRY = (0:DIVIDE_ERROR,</td>
<td>1:SINGLE_STEP,</td>
</tr>
<tr>
<td>2:NMI,</td>
<td>3:BREAKPOINT,</td>
</tr>
<tr>
<td>4:INTO_OVERFLOW,</td>
<td>5:BOUND_RANGE,</td>
</tr>
<tr>
<td>6:INVALID_OPCODE,</td>
<td>7:PROC_EXT_NOT_AVAILABLE,</td>
</tr>
<tr>
<td>8:DOUBLE_EXCEPTION,</td>
<td>9:PROC_EXT_SEGMENT_OVERRUN,</td>
</tr>
<tr>
<td>10:INVALID_TASK,</td>
<td>11:SEGMENT_NOT_PRESENT,</td>
</tr>
<tr>
<td>12:STACK_OVERRUN,</td>
<td>13:GENERAL_PROTECTION,</td>
</tr>
<tr>
<td>16:NCP_INTERRUPT,</td>
<td>80:CHA_TX_EMPTY,</td>
</tr>
<tr>
<td>82:CHB_RX_AVAIL,</td>
<td>83:CHA_RX_ERROR,</td>
</tr>
<tr>
<td>84:CHA_TX_EMPTY,</td>
<td>86:CHA_RX_AVAIL,</td>
</tr>
<tr>
<td>87:CHA_RX_ERROR,</td>
<td></td>
</tr>
</tbody>
</table>
The BNOP2.BAT batch file was written in order to bind the 80286 protected mode code associated with the secondary processor to the standardised code. This batch file generated the following command file:

```
BNOP2 PM_PROC2.OBJ, &
PM_INIT.OBJ, &
PM_INTS.OBJ, &
PM_SEGS.OBJ, &
STDCPU.OBJ, &
STDRAM.OBJ, &
STDERAM.OBJ, &
STDPIC.OBJ, &
STDSTAT.OBJ, &
STDBIT.OBJ, &
STDINTS.OBJ, &
STDIO.OBJ, &
STDCONVT.OBJ, &
NOLOAD &
NAME (PM_PROC2) &
OBJECT (PM_PROC2)
```
3.7 BUILDING THE SECONDARY PROCESSOR PROTECTED MODE CODE

In a similar manner to the primary processor code, the output of the BNDP2 batch file was bound to the module to enter protected mode and located by means of a build file. The command and build files are included below:

```plaintext
BLD286 PM_Entp.OBJ, &
    PM_PROC2 &
    BF(PROC2.BLD) &
    OBJECT(PM_PROC2.LOC)
EXIT
CTTY CON

PROC2_BLD;

SEGMENT

*SEGMENTS (DPL = 0),

ENTER_PROTECTED_MODE_CODE
(DPL = 0, BASE = 0AFF00H),

ENTER_PROTECTED_MODE_STACK
(DPL = 0, PRESENT,
BASE = 0E0000H,
LIMIT = 1FFFFH, ED),

PM_PROC2_STACK
(DPL = 0, PRESENT,
BASE = 0E0000H,
LIMIT = 1FFFFH, ED),

BIT_RESULTS_SEG
(DPL = 0, BASE = 5000H),

TEST_RESULTS_SEG
(DPL = 0, BASE = 5000H),

PROC_ID_SEG
(DPL = 0, BASE = 5200H),

SLAVE_SEG
(DPL = 0, BASE = 5280H),

GLOBAL_RAM
(DPL = 0, BASE = 5000H),

PROC2_GLOBAL_SEG
(DPL = 0, BASE = 11220H),

API_SEG_1
(DPL = 0, BASE = 11000H),

API_SEG_2
(DPL = 0, BASE = 11000H),

SDB_BUFFER
(DPL = 0, BASE = 20800H),

MAP_SLAVE_SEG
(DPL = 0, BASE = 40000H),

RAM_BLOCK
(DPL = 0, BASE = 0H, LIMIT = OFFFFH),

ERAM_BLOCK_1
(DPL = 0, BASE = 40000H, LIMIT = 0FFFFH),

ERAM_BLOCK_2
(DPL = 0, BASE = 50000H, LIMIT = 0FFFFH),

ERAM_BLOCK_3
(DPL = 0, BASE = 60000H, LIMIT = 0FFFFH),

ERAM_BLOCK_4
(DPL = 0, BASE = 70000H, LIMIT = 0FFFFH),

ERAM_BLOCK_5
(DPL = 0, BASE = 80000H, LIMIT = 0FFFFH),
```
EROM_BLOCK_1
(DPL = 0, BASE = 60000H, LIMIT = 0FFFFH),
EROM_BLOCK_2
(DPL = 0, BASE = 70000H, LIMIT = 0FFFFH),
EROM_BLOCK_3
(DPL = 0, BASE = 80000H, LIMIT = 0FFFFH),
EROM_BLOCK_4
(DPL = 0, BASE = 90000H, LIMIT = 0FFFFH),
EROM_BLOCK_5
(DPL = 0, BASE = 0A0000H, LIMIT = 0FFFFH),
EROM_BLOCK_6
(DPL = 0, BASE = 0B0000H, LIMIT = 0FFFFH),
EROM_BLOCK_7
(DPL = 0, BASE = 0C0000H, LIMIT = 0FFFFH),
EROM_BLOCK_8
(DPL = 0, BASE = 0D0000H, LIMIT = 0FFFFH),
SYSTEM_DESC
(DPL = 0, BASE = 0D0000H, LIMIT = 0FFFFH),
EROM_BLOCK_9
(DPL = 0, BASE = 0E0000H, LIMIT = 0FFFFH),
EROM_BLOCK_10
(DPL = 0, BASE = 0F0000H, LIMIT = 0FFFFH),
DRAM_LBUS_SEG
(DPL = 0, BASE = 10000H),
DRAM_MBUS_SEG
(DPL = 0, BASE = 30000H),
PROC2_SLAVE_SEG
(DPL = 0, BASE = 36000H),
SLAVE_BUFFER_SEG
(DPL = 0, BASE = 350000H);

TASK

TASK_1 (OBJECT = ENTER_PROTECTED_MODE,
DPL = 0,
BASE = 1COOH,
LIMIT = 100,
NOT INTENABLE);

SYSTEM_EXECUTIVE (OBJECT = PM_PROC2,
DPL = 0,
BASE = 1D00H,
LIMIT = 100,
STACKS = (PM_PROC2.STACK),
NOT INTENABLE);

GATE

SYSTEM_EXECUTIVE (TASK, ENTRY = SYSTEM_EXECUTIVE);
TABLE
LDT_2 (DPL = 0, PRESENT, BASE = 0C00H, LIMIT = 100,
ENTRY = (99:SYSTEM_EXECUTIVE)
);

GATE
DIVIDE_ERROR(INTERUPT), SINGLE_STEP(INTERUPT),
NMI(INTERUPT), BREAKPOINT(INTERUPT),
INTO_OVERFLOW(INTERUPT), BOUND_RANGE(INTERUPT),
INVALID_OPCODE(INTERUPT), PROC_EXT_NOT_AVAILABLE(INTERUPT),
DOUBLE_EXCEPTION(INTERUPT), PROC_EXT_SEGMENT_OVERRUN(INTERUPT),
INVALID_TASK(INTERUPT), SEGMENT_NOT_PRESENT(INTERUPT),
STACK_OVERRUN(INTERUPT), GENERAL_PROTECTION(INTERUPT),
NCP_INTERRUPT(INTERUPT), CHA_TX_EMPTY(INTERUPT),
CHB_TX_EMPTY(INTERUPT), CHA_RX_AVAIL(INTERUPT),
CHB_RX_AVAIL(INTERUPT), CHA_RX_ERROR(INTERUPT),
CHB_RX_ERROR(INTERUPT),

TMOUT(INTERUPT), REAL_TIME_CLOCK(INTERUPT),
MASTER_INTERRUPT_2(INTERUPT), MASTER_INTERRUPT_3(INTERUPT),
MASTER_INTERRUPT_4(INTERUPT), MASTER_INTERRUPT_7(INTERUPT),

ILLEGAL_INT(INTERUPT),

API(INTERUPT), SCMB(INTERUPT),
SLAVE_INTERRUPT_2(INTERUPT), PROC2_INTERRUPT(INTERUPT),
SLAVE_CLOCK(INTERUPT), SLAVE_INTERRUPT_5(INTERUPT),
SLAVE_INTERRUPT_6(INTERUPT), SLAVE_INTERRUPT_7(INTERUPT)
;

TABLE
GDT (DPL = 0, PRESENT, BASE = 800H, LIMIT = 100, ENTRY = (LDT_2)),
IDT (DPL = 0, PRESENT, BASE = 0H, LIMIT = 256,
ENTRY = (0:DIVIDE_ERROR, 1:SINGLE_STEP,
2:NMI, 3:BREAKPOINT,
4:INTO_OVERFLOW, 5:BOUND_RANGE,
6:INVALID_OPCODE, 7:PROC_EXT_NOT_AVAILABLE,
8:DOUBLE_EXCEPTION, 9:PROC_EXT_SEGMENT_OVERRUN,
10:INVALID_TASK, 11:SEGMENT_NOT_PRESENT,
12:STACK_OVERRUN, 13:GENERAL_PROTECTION,
16:NCP_INTERRUPT, 80:CHA_TX_EMPTY,
82:CHB_RX_AVAIL, 83:CHA_RX_ERROR,
84:CHA_TX_EMPTY, 86:CHA_RX_AVAIL,
87:CHA_RX_ERROR,
96:TMOUT, 97:REAL_TIME_CLOCK,
98:MASTER_INTERRUPT_2, 99:MASTER_INTERRUPT_3,
100:MASTER_INTERRUPT_4, 103:MASTER_INTERRUPT_7,
3.8 PREPARING THE PROTECTED MODE CODE FOR THE EPROMS

The preparation of the protected mode code for the EPROMS on the EPROM/RAM cards involved:

(a) translating the 80286 object module files output by the builder to 8086 object module files before converting these files to hexadecimal.

(b) extracting the descriptor tables from the hexadecimal files in order to absolutely locate them prior to binary conversion.

Step (a) was achieved by means of an Intel-supplied software package, known as IPPS. This package effectively transformed the file header to that of an 8086 object module format and the hexadecimal conversion then took place through the 8086 object to hexadecimal converter in the normal manner.

The extraction of the descriptor tables, on the other hand, was accomplished by editing the two protected mode hexadecimal files and splitting each of them into two files, one containing the protected mode code and the other containing the descriptor tables. The binary file creation was then a simple matter of passing each file through the binary converter and mapping the addresses according to the memory map of Figure 4.

3.9 EPROM LOADING PROCEDURE FOR THE SUBSYSTEM-SPECIFIC CODE

The "HI-LO" EPROM programmer unit [INTEL, 1987] was also used to program the EPROMS resident on the EPROM/RAM cards. Following invocation of the Intel EPROM programming software, the following procedures ensued:

1) The programmer was initialised to the correct manufacturer and chip.

2) The low address EPROMS (i.e. those that were to contain the protected mode code) were programmed directly with the primary and secondary processor binary files.
3) The high address EPROMS were programmed with the real mode code, the two descriptor tables and the appropriate values for the system description.

4) The EPROMS were inserted into the appropriate sockets on the EPROM/RAM card associated with the primary processor and the cards inserted into the system.

5) The system was powered up and after completion of the system POST, the calculated checksum was noted.

6) Finally, the system was powered down, the EPROMS extracted and then re-programmed with the appropriate checksums.
4 REFERENCES

[INTEL, 1987]
# APPENDIX E: OPERATIONAL DEMONSTRATION PROCEDURE

## 1 INTRODUCTION

## 2 PRELIMINARY POWER UP

## 3 POWER UP

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 Normal Console Power Up</td>
<td>7</td>
</tr>
<tr>
<td>3.2 Normal System Power Up</td>
<td>8</td>
</tr>
<tr>
<td>3.3 Fault Detection During Power Up</td>
<td>9</td>
</tr>
<tr>
<td>3.3.1 Console fault detection</td>
<td>9</td>
</tr>
<tr>
<td>3.3.2 System fault detection</td>
<td>15</td>
</tr>
</tbody>
</table>

## 4 On-Line Console Functional Demonstrations

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 Rollerball Module Functional Demonstration</td>
<td>40</td>
</tr>
<tr>
<td>4.2 Graphics Display Unit Demonstration</td>
<td>41</td>
</tr>
<tr>
<td>4.3 Multimaster Bus Contention Demonstration</td>
<td>43</td>
</tr>
<tr>
<td>4.4 Local Area Network Functional Demonstration</td>
<td>44</td>
</tr>
<tr>
<td>4.4.1 Normal operation</td>
<td>44</td>
</tr>
<tr>
<td>4.4.2 Local area network fault detection</td>
<td>47</td>
</tr>
<tr>
<td>4.5 System Reset Demonstration</td>
<td>48</td>
</tr>
<tr>
<td>4.6 Applications Processor Interface Functional Demonstration</td>
<td>49</td>
</tr>
<tr>
<td>4.7 Manual Input Devices Functional Demonstration</td>
<td>50</td>
</tr>
<tr>
<td>4.7.1 Softkey modules functional demonstration</td>
<td>53</td>
</tr>
<tr>
<td>4.7.2 &quot;Qwerty&quot; keyboard module functional demonstration</td>
<td>54</td>
</tr>
<tr>
<td>4.8 Environmental Monitoring and Control Functional Demonstration</td>
<td>55</td>
</tr>
<tr>
<td>4.8.1 Environmental monitoring and control normal operation</td>
<td>55</td>
</tr>
<tr>
<td>4.8.2 Environmental monitoring and control fault detection</td>
<td>59</td>
</tr>
<tr>
<td>4.9 System Override Functional Demonstration</td>
<td>61</td>
</tr>
<tr>
<td>4.10 Interlock Override Functional Demonstration</td>
<td>62</td>
</tr>
<tr>
<td>4.11 Mass Storage Controller Demonstration</td>
<td>63</td>
</tr>
<tr>
<td>4.11.1 Mass storage controller normal operation</td>
<td>63</td>
</tr>
<tr>
<td>4.11.2 Mass storage controller fault detection</td>
<td>68</td>
</tr>
</tbody>
</table>
5 OFF-LINE DIAGNOSTICS

5.1 PRIMARY PROCESSOR TESTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1.1 Primary processor self-test</td>
<td>72</td>
</tr>
<tr>
<td>5.1.2 Primary processor EPROM/RAM card test</td>
<td>72</td>
</tr>
<tr>
<td>5.1.3 Primary processor dynamic RAM card test</td>
<td>73</td>
</tr>
<tr>
<td>5.1.4 Primary processor graphics interface modules test</td>
<td>74</td>
</tr>
<tr>
<td>5.1.5 Primary processor serial communications to Multibus card test</td>
<td>75</td>
</tr>
<tr>
<td>5.1.6 Primary processor applications processor interface card test</td>
<td>75</td>
</tr>
<tr>
<td>5.1.7 Primary processor system data bus controller card test</td>
<td>76</td>
</tr>
<tr>
<td>5.1.8 Primary processor mass storage controller card test</td>
<td>77</td>
</tr>
<tr>
<td>5.1.9 Primary processor test on the secondary processor</td>
<td>78</td>
</tr>
</tbody>
</table>

5.2 SECONDARY PROCESSOR TESTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2.1 Secondary processor self-test</td>
<td>79</td>
</tr>
<tr>
<td>5.2.2 Secondary processor EPROM/RAM test</td>
<td>80</td>
</tr>
<tr>
<td>5.2.3 Secondary processor dynamic RAM test</td>
<td>81</td>
</tr>
<tr>
<td>5.2.4 Secondary processor graphics interface modules test</td>
<td>82</td>
</tr>
<tr>
<td>5.2.5 The secondary processor serial communications to Multibus card test</td>
<td>83</td>
</tr>
<tr>
<td>5.2.6 Secondary processor applications processor interface card test</td>
<td>84</td>
</tr>
<tr>
<td>5.2.7 Secondary processor system data bus controller card test</td>
<td>85</td>
</tr>
<tr>
<td>5.2.8 Secondary processor mass storage controller card test</td>
<td>85</td>
</tr>
</tbody>
</table>

6 REFERENCES
LIST OF TABLES

Table I : Main demonstration menu 39
Table II : Rollerball demonstration menu 40
Table III : Graphics demonstration menus 41
Table IV : Graphics display unit selection menu 42
Table V : Serial communications to Multibus menu 44
Table VI : High-level data link control menu 46
Table VII : HDLC baud rate selection menu 46
Table VIII : Real mode demonstration menu 48
Table IX : Console demonstration menu 49
Table X : Applications processor interface demonstration menu 49
Table XI : Manual input devices demonstration menu 51
Table XII : Softkey module demonstration menu 52
Table XIII : "Qwerty" keyboard demonstration menu 52
Table XIV : Environmental monitoring and control menu 52
Table XV : Console status request menu 56
Table XVI : Further status request menus 57
Table XVII : Control mode menu 58
Table XVIII : Mass storage controller card demonstration menu 63
Table XIX : Mass storage controller card fault simulation menu 69
Table XX : Primary processor off-line diagnostics menu 71
Table XXI : Secondary processor off-line diagnostic menu 78
This appendix describes the operational demonstration procedures executed at UEC Projects (Pty.) Ltd. that prove the functional operation of the real-time embedded system [LAW-BROWN, 1990]. The tests and demonstrations built into the development system cover both the normal fault-free power up and the cases where faults are in existence at the time of power up.

In particular, the following areas of functionality are presented:

(a) Power up  
(b) On-line console functional demonstrations  
(c) Off-line diagnostics facility

In order to execute the test procedures described, the following support equipment was required:

(a) an RS232 terminal (keyboard and screen)  
(b) a multimeter  
(c) a 100MHz oscilloscope  
(d) a 220V 50Hz Mains Supply  
(e) a logic analyzer

The on-line console functional demonstrations and the off-line diagnostics demonstrations were presented in menu form and were selected via the RS232 terminal. The menu options were displayed on the RS232 terminal connected to the diagnostic serial port located on the control and status panel. Test results were also displayed on the RS232 terminal.
2 PRELIMINARY POWER UP

Prior to inserting the cards into the card cage, the power supplies were checked. The following steps were performed:

(a) The mains lead was plugged into a suitable 15A 220V socket and power applied to the console. The green pilot light (mains available) mounted on the control and status panel module illuminated to indicate that power had been connected to the console.

(b) The circuit breaker mounted on the gland and terminal housing unit was moved to the "on" position and the power switch pressed. This resulted in power being fed to the cardcage.

(c) The interlock override switch on the control and status panel module was pressed and the cardcage withdrawn.

(d) A multimeter was used to check that the following voltages appeared at the corresponding tagblocks on the back of the cardcage:

\[
\begin{align*}
+5V & \quad +/\ - \quad 0.1V \\
+12V & \quad +/\ - \quad 0.2V \\
-12V & \quad +/\ - \quad 0.2V
\end{align*}
\]

\(\text{(tagblock 2 & tagblock 5)}\) \\
\(\text{(tagblock 1)}\) \\
\(\text{(tagblock 4)}\)

(e) Once the correct values had been observed the console was powered down and the required cards inserted. The RS232 terminal was then connected to the diagnostics port on the control and status panel module.
3 POWER UP

The power up results were described in the chapter that dealt with the software development. A recapitulation is included here for clarity.

During the power up routine the main applications processor does the following:

(a) tests the microprocessor, on-board ROM and on-board RAM
(b) tests the on-board peripherals
(c) tests its off-board access ability
(d) tests the EPROM/RAM card
(e) tests for the presence and status of the system data bus controller card
(f) tests the local bus connection
(g) tests for the presence and status of the applications processor interface card
(h) tests for the presence and status of the first graphics card
(i) tests for the presence and status of the second graphics card
(j) tests for the presence and status of the third graphics card
(k) tests for the presence and status of the serial communications to Multibus card
(l) tests for the presence and status of the mass storage controller card
(m) tests the dynamic RAM card
(n) tests for the presence and status of the secondary processor

The results of these power up tests are recorded in an array in memory. The execution of these tests were observed by monitoring the seven segment displays on the bus terminator unit card. Tests (a) to (f) constitute the standard computing segment POST. Any failure occurring before test (h) was not reported to any of the graphics display units since access to the graphics cards was unproven. If possible, the main applications processor reported to the RS232 terminal connected to the diagnostics port. However, the graphics interface modules reported the failure of the main applications processor to poll them within a predefined timeout period.

The main applications processor reported any failures or missing cards detected subsequent to test (h) on the first graphics display unit available as well as the RS232 terminal screen. If all the power up tests passed, successful operation was indicated by the display of the demonstration menu on the
RS232 terminal and the replacement of the test pattern on the first graphics display unit with the same demonstration menu. The other two graphics display units displayed a power up test pattern to visually indicate the correct functioning of the graphics cards and associated displays.

Operational demonstration tests of this power up routine consisted of powering up the system and observing the main demonstration menu on the first graphics display unit. Non-critical faults or failures detected resulted in a fault report on the graphics display unit and progression to the main demonstration menu was possible.

The tests listed below were intended to prove the correct performance of all of the functions forming the power up of the console. The console was initially configured with a full compliment of cards in the card cage.

3.1 NORMAL CONSOLE POWER UP

(a) Test Procedure

Power was applied to the mains lead and the circuit breaker on the gland and terminal housing unit moved to the "on" position.

(b) Result

(i) The backlighting of the control and status panel module switches occurred.
(ii) The system override indicated "off".
(iii) Power indicators on the graphics display units and the card cage remained off.
(iv) The AP, API, PROC and MIDS fault LEDs located on the control and status panel module were illuminated.

(c) Result Interpretation

The backlighting of the lamps and the illumination of the AP, API, PROC and MIDS fault LEDs indicated the correct operation of the environmental monitoring and control circuitry.
3.2 NORMAL SYSTEM POWER UP

(a) Test Procedure

Power was applied to the console, the circuit breaker on the gland and terminal housing unit moved to the "on" position, and the console switched on.

(b) Result

(i) The switches of the control and status panel module were illuminated.
(ii) The power indicators on the graphics display units and the card cage were illuminated.
(iii) The PROC fault LED flashed repeatedly.
(iv) Test logos were displayed on the graphics display units.
(v) The AP, API and MIDS LEDs extinguished after approximately two minutes.

(c) Result Interpretation

The flashing of the PROC LED indicated the commencement of the POST routines by the processors in the card cage. The AP, API and MIDS fault LEDs were extinguished when the processor completed the test of the standard computing segment and initialised the applications processor interface. Successful completion of the system POST was indicated by the PROC fault LED being extinguished and the display of the main demonstration menu on the graphics display unit.
3.3 FAULT DETECTION DURING POWER UP

The ability of the system to detect faults during the power up phase of operation was tested in this portion of the demonstration procedure. After simulating each fault to demonstrate the fault detection capabilities at power up, the system and console were powered down and the system returned to the normal operational state.

3.3.1 Console fault detection

The following tests were conducted on the system to validate the ability of the BIT/BITE to detect a failure or abnormal status at console level.

3.3.1.1 Faulty environmental monitoring and control card

(a) Fault simulation

(i) In order to simulate a failure of the environmental monitoring and control card, the card was removed from its card cage position.

(ii) The circuit breaker was moved to the "on" position in order to apply power to the console.

(b) Result

(i) There was no backlighting of the control and status panel module switches.

(ii) The AP, API and MIDS fault LEDs were not illuminated.

(iii) The power indicators for the graphics display units and the card cage were immediately illuminated.

(iv) The PROC fault LED began to flash.

(v) Test logos appeared on the graphics display units while the processor performed its power on tests.

(vi) A fault report indicating the environmental monitoring and control card absent or faulty was displayed on the first graphics display unit.

(c) Result Interpretation

Failure or absence of the environmental monitoring and control resulted in power being applied immediately to all of the units in the console. The main applications
processor detected the failure of the environmental monitoring and control card during the power up tests and generated the fault report.

3.3.1.2 Failure of the environmental monitoring and control to applications processor interface link

(a) Fault simulation

(i) Plug twelve on the I/O connector plate of the third card cage was disconnected to simulate a failure of the serial link between the environmental monitoring and control card and the applications processor interface.

(ii) The circuit breaker was moved to the "on" position.

(iii) The console was powered up.

(b) Result

(i) Step (ii) above resulted in backlighting of the switches on the control and status panel module and the illumination of the AP, API, and MIDS fault LEDs. Power was not applied to the graphics display units and the card cage.

(ii) Step (iii) resulted in power being applied to the graphics display units and the card cage allowing the main applications processor to run. Normal power up continued resulting in a report of the environmental monitoring and control fault. The fault LEDs on the control and status panel module were not extinguished. Operation could be continued by striking any key.

(c) Result Interpretation

Due to the broken link between the environmental monitoring and control card and the applications processor interface, the status of the environmental monitoring and control and any device connected to it could not be determined by the applications processor. In addition, the applications processor could not reset the fault LEDs on the control and status panel module. The PROC fault LED was, however, controlled directly by the applications processor via an independent link and this continued to display the correct status.
3.3.1.3 Detection of abnormal switch status on control and status panel module

(a) Fault simulation

(i) One of the switches provided on the control and status panel module which would not normally be operated at power up was pressed.
(ii) The system was powered up.
(iii) After the results were recorded, the system was powered down and the previous steps repeated for various combinations of switches which would not normally be operated at power up.

(b) Result

Normal power up was completed and resulted in the display of a fault report detailing the unexpected switches which were closed.

(c) Result Interpretation

The ability of the applications processor to determine the console switch status at power up or subsequently was demonstrated.
3.3.1.4 Detection of stuck key on any softkey module

(a) Fault simulation

(i) One of the softkeys was pressed.
(ii) The system was powered up.
(iii) After the results were recorded, the system was powered down and the previous steps repeated for various combinations of switches which would not normally be operated at power up.

(b) Result

Normal power up was completed and resulted in the display of a fault report detailing which keys were stuck.

(c) Result Interpretation

The ability of the applications processor to determine key status during power up or subsequently was demonstrated.
3.3.1.5 Detection of incorrectly closed drawer housing the first graphics display unit

(a) Fault simulation

(i) The thumb screws securing the first graphics display unit drawer were undone and the drawer partially withdrawn.

(ii) The console and the system were powered up.

(b) Result

(i) Normal power up proceeded without power indication on the partially withdrawn drawer. This procedure resulted in the generation of a fault report, displayed on the second graphics display unit, identifying the unclosed drawer.

(c) Result Interpretation

The identification of an incorrectly closed drawer housing the first graphics display unit was achieved. The power indicator on the drawer and operation of the interlock facility were also validated.
3.3.1.6 Detection of incorrectly closed drawers housing the other graphics display units and the second card cage

(a) Fault simulation

(i) The thumb screws securing the second graphics display unit drawer were released and the drawer partially withdrawn.
(ii) The console and the system were powered up.
(iii) The procedure was repeated for the third graphics display unit drawer and the drawer housing the second card cage.

(b) Result

(i) Normal power up proceeded without power indication on the partially withdrawn drawer. The generation of a fault report resulted, identifying the respective unclosed drawer.

(c) Result Interpretation

The identification of an incorrectly closed drawer housing the second and third graphics display units and the second card cage was achieved. The power indicator on the respective drawer and operation of the respective interlock facility were also validated.
3.3.1.7 Detection of incorrectly closed drawer housing the first card cage

(a) Fault simulation

(i) The thumb screws securing the drawer housing the first card cage were released and the drawer partially withdrawn.
(ii) The console and the system were powered up.

(b) Result

(i) Powering up the system caused the PROC fault LED to illuminate and not flash.
(ii) The AP, API and MIDS fault LEDs were not extinguished.
(iii) Power was not applied to the first card cage.

(c) Result Interpretation

Incorrect closure of the applications processor card cage or the failure of the interlock on the drawer was detected.

3.3.2 System fault detection

The following tests were conducted on the system to validate the ability of the POST to detect failure or absence of any of the cards in the applications processor.

For all of the tests listed below the system was configured with the cards fitted in their appropriate slots as indicated in the processor card cage layout diagram presented earlier. In cases where faults have been simulated, any changes to this normal configuration are indicated.

Recall that the implemented philosophy was for critical errors to result in the halting of the detecting processor, leaving a display of the test number which failed on the relevant seven segment display. During operational demonstration testing, the primary processor also displayed an error message on the RS232 terminal if this was possible.
3.3.2.1 No Multibus clocks

(a) Fault simulation

(i) The primary CPU card was withdrawn from the card cage.
(ii) The console and the system were powered up.

(b) Result

(i) Normal power up appeared to commence as the PROC fault LED began to flash and the test logos appeared on the screens.
(ii) After the time-out period of the graphics cards the test logos were replaced by a fault message reporting the failure of both processors to run.
(iii) The seven segment LED display associated with main applications processor displayed the error code FF hexadecimal.
(iv) The seven segment LED display associated with the secondary processor displayed the error code 40 hexadecimal.
(v) Both the BCLOCK and CCLOCK fault LEDs on the bus terminator unit were illuminated.
(vi) The API and MIDS fault LEDs were extinguished.

(c) Result Interpretation

The primary CPU card provided the clocks for the Multibus. Failure of the Multibus was detected by the secondary processor resulting in the generation of the error code 40 hexadecimal. The graphics cards were not initialised by either of the processors and timed-out, resulting in the generation of the fault reports on all the screens. The BITE on the bus terminator unit successfully monitored the Multibus clocks and provided the expected fault indication.
3.3.2.2 Primary processor failure

(a) Fault simulation

(i) The EPROMS were removed from the primary CPU card in order to simulate its failure and the card returned to its position in the card cage.

(ii) The console and the system were powered up.

(b) Result

(i) Normal power up appeared to commence.

(ii) A fault report appeared on the first graphics display unit indicating that the secondary processor had taken over as the main applications processor.

(iii) Advancement to the main demonstration menu as well as the execution of demonstrations not requiring both processors was possible.

(c) Result Interpretation

The secondary processor detected a failure of the primary processor to rendezvous. The former replaced the latter and executed as the main applications processor in a degraded fashion.
3.3.2.3 Secondary processor failure

(a) Fault simulation

(i) The secondary processor was removed from the card cage in order to simulate its failure.

(ii) The console and the system were powered up.

(b) Result

(i) Normal power up appeared to commence.

(ii) A fault report was displayed on the first graphics display unit indicating the absence of the secondary processor.

(iii) Advancement to the main demonstration menu was possible.

(c) Result Interpretation

The failure of the secondary processor was detected and the system was able to continue with degraded performance.
3.3.2.4 Failure of the primary processor local bus

(a) Fault simulation

(i) The third graphics interface module was removed from its slot and the primary EPROM/RAM card inserted into this position. This simulated a failure of the local bus connection between the primary processor and the primary EPROM/RAM card.

(ii) The console and the system were powered up.

(b) Result

(i) Normal power up commenced.

(ii) The test logo on the first graphics display unit was replaced by a fault report indicating the failure of the local bus connection to the primary EPROM/RAM card and the absence of the third graphics card.

(iii) Advancement to the main demonstration menu was possible.

(c) Result Interpretation

The ability of the POST to detect the failure of the local bus connection as well as the ability of the system to continue operation via the Multibus was demonstrated.
3.3.2.5 Faulty ROM on the primary EPROM/RAM card

(a) Fault simulation

(i) One of the EPROMS was removed from the primary EPROM/RAM card in order to simulate a faulty EPROM and the card returned to its slot.

(ii) The console and the system were powered up.

(b) Result

(i) Normal power up appeared to commence.

(ii) The RS232 terminal displayed a checksum error message if the removed EPROM was one of the lower EPROMS or the "EPROM/RAM card initialization screen" if one of the higher EPROMS was removed. The test logo on the first graphics display unit was replaced by a fault report indicating that the secondary processor had taken over as the main applications processor.

(iii) The seven segment LED display associated with the primary CPU card displayed the error code 11 hexadecimal.

(iv) Advancement to the main demonstration menu was possible.

(c) Result Interpretation

The error code 11 hexadecimal generated by the primary processor indicated the detection of a fault on the primary EPROM/RAM card.
3.3.2.6 Faulty RAM on the primary EPROM/RAM card

(a) Fault simulation

(i) One of the RAM chips was removed from the primary EPROM/RAM card in order to simulate faulty RAM and the card was returned to its slot.
(ii) The console and the system were powered up.

(b) Result

(i) Normal power up appeared to commence.
(ii) The RS232 terminal displayed an error message indicating the absence or failure of the primary EPROM/RAM card. The test logo on the first graphics display unit was replaced by a fault report indicating that the secondary processor had taken over as the main applications processor.
(iii) The seven segment LED display associated with the primary CPU card displayed the error code 10 hexadecimal.
(iv) Advancement to the main demonstration menu was possible.

(c) Result Interpretation

The error code 10 hexadecimal generated by the primary processor indicated the detection of an absent or faulty primary EPROM/RAM card.
3.3.2.7 Absent primary EPROM/RAM card

(a) Fault simulation

(i) The primary EPROM/RAM card was removed from the card cage.
(ii) The console and the system were powered up.

(b) Result

(i) Normal power up appeared to commence.
(ii) The RS232 terminal displayed an error message indicating the absence or failure of the primary EPROM/RAM card. The test logo on the first graphics display unit was replaced by a fault report indicating that the secondary processor had taken over as the main applications processor.
(iii) The seven segment LED display associated with the primary CPU card displayed the error code 10 hexadecimal.
(iv) Advancement to the main demonstration menu was possible.

(c) Result Interpretation

The error code 10 hexadecimal generated by the primary processor indicated the detection of an absent or faulty primary EPROM/RAM card.
3.3.2.8 Failure of the secondary processor local bus

(a) Fault simulation

(i) The secondary EPROM/RAM card was removed from its slot simulating a failure of the secondary processor local bus.
(ii) The console and the system were powered up.

(b) Result

(i) Normal power up commenced.
(ii) The test logo on the first graphics display units was replaced by a fault message indicating the failure of the local bus link between the secondary processor and the secondary EPROM/RAM card.
(iii) Advancement to the main demonstration menu was possible.

(c) Result Interpretation

Due to the absence of the secondary EPROM/RAM card, the secondary processor defaults to Multibus and executes code located on the primary EPROM/RAM card. The Multibus access was much slower and the code therefore detected a failure of the secondary processor local bus. The secondary local bus was the only link that the secondary EPROM/RAM card had with the system. Thus it was not possible for the secondary processor to distinguish between a local bus failure or the absence of its associated EPROM/RAM card. Due to the system architecture, either of these faults will always be interpreted as a failure of the secondary processor local bus.
3.3.2.9 Faulty ROM on the secondary EPROM/RAM card

(a) Fault simulation

(i) One of the EPROMS was removed from the secondary EPROM/RAM card in order to simulate a faulty EPROM and the card returned to its slot.

(ii) The console and the system were powered up.

(b) Result

(i) Normal power up appeared to commence.

(ii) The test logo on the first graphics display unit was replaced by a fault report indicating that the secondary processor was absent.

(iii) The seven segment LED display associated with the secondary processor displayed the error code 11 hexadecimal.

(iv) Advancement to the main demonstration menu was possible.

(c) Result Interpretation

The error code 11 generated by the secondary processor indicated the detection of the faulty ROM on the secondary EPROM/RAM card.
3.3.2.10 Faulty RAM on the secondary EPROM/RAM card

(a) Fault simulation

(i) One of the RAM chips was removed from the secondary EPROM/RAM card in order to simulate a faulty RAM and the card returned to its slot.

(ii) The console and the system were powered up.

(b) Result

(i) Normal power up appeared to commence.

(ii) The test logo on the first graphics display unit was replaced by a fault report indicating that the secondary processor was absent.

(iii) The seven segment LED display associated with the secondary processor displayed the error code 10 hexadecimal.

(iv) Advancement to the main demonstration menu was possible.

(c) Result Interpretation

The error code 10 generated by the secondary processor indicated the detection of the faulty RAM on the secondary EPROM/RAM card.
3.3.2.11 Absent applications processor interface

(a) Fault simulation

(i) The applications processor interface card was removed from its slot.
(ii) The console and the system were powered up.

(b) Result

(i) Normal power up commenced.
(ii) The test logos displayed on all three of the graphics display units was replaced by an error display indicating the failure of both processors to complete the POST.
(iii) Both status latches associated with each CPU card displayed error code 40 indicating the failure/absence of the applications processor interface card.
(iv) The RS232 terminal displayed an error message indicating the failure/absence of the applications processor interface.

(c) Result Interpretation

The POST detected the failure/absence of the applications processor interface card. As the availability of the dual port RAM on this card was essential for the dual processor rendezvous, operation of both processors was stopped when an absent card was detected.
3.3.2.12 Failure of dynamic RAM on the local bus

(a) Fault simulation

(i) The third graphics interface module was removed from its slot and the dynamic RAM card inserted into the vacant slot, thus simulating a failure of the local bus link.

(ii) The console and the system were powered up.

(b) Result

(i) Normal power up commenced.

(ii) The test logo on the first graphics unit was replaced by a fault message indicating the failure of the local bus link between the main applications processor and the dynamic RAM card as well as the absence of the third graphics card.

(iii) Advancement to the main demonstration menu was possible.

(c) Result Interpretation

The primary CPU card detected the inability to access local bus dynamic RAM addresses.
3.3.2.13 Failure of the dynamic RAM card via the Multibus

(a) Fault simulation

(i) The dynamic RAM card was connected to an extender card and the combination re-inserted into the dynamic RAM card slot. The transfer acknowledge link (i.e. XACK link 10b) of the upper (P1) connector was removed in order to simulate the failure of the Multibus link.

(ii) The console and the system were powered up.

(b) Result

(i) Normal power up commenced.

(ii) The test logo on the first graphics unit was replaced by a fault message indicating the failure of the dynamic RAM card. The RS232 terminal displayed a message indicating that a critical error had occurred and that the system had been halted. The status latch displayed the error code 70 hexadecimal.

(iii) Advancement to the main demonstration menu was not possible.

(c) Result Interpretation

Error messages were generated by the graphics cards as a result of commands issued to their general purpose registers. This allowed the error status of the dynamic RAM to be displayed on the graphics display unit. However, non-error messages and other screen displays were generated in response to command strings placed in dynamic RAM for retrieval by the graphics cards. This ability was lost due to the non-existent memory. For this reason, the test was defined as critical and, on detection of the failure to communicate via Multibus, the system was consequently shut down.
3.3.2.14 Absence of the dynamic RAM card

(a) Fault simulation

(i) The dynamic RAM card was removed from the card cage.
(ii) The console and the system were powered up.

(b) Result

(i) Normal power up commenced.
(ii) The test logo on the first graphics unit was replaced by an error message indicating the failure of the dynamic RAM card. The RS232 terminal displayed an error message indicating that a critical error had been detected and the system had been shut down. The error code 70 hexadecimal was displayed on the status latch.
(iii) Further advancement was not possible.

(c) Result Interpretation

The absence of the dynamic RAM was not established because the Multibus access to the card was the first test to be conducted. Thus the same results were generated as those of a failure to communicate over the Multibus.
3.3.2.15 First graphics card unable to report its status

(a) Fault simulation

(i) The microprocessor of the first graphics card was removed in order to simulate a failure of the card and the card returned to its slot.

(ii) The console and the system were powered up.

(b) Result

(i) Normal power up commenced.

(ii) The first graphics unit did not display a test logo.

(iii) The test logo displayed on the second graphics display unit was replaced by a fault report indicating that the first graphics interface module failed to report its status.

(iv) Advancement to the main demonstration menu (displayed on the second graphics display unit) was possible.

(c) Result Interpretation

The POST detected the failure of the first graphics card to report its status and redirected the fault display to the second graphics display unit via the associated graphics interface module.
3.3.2.16 Second and third graphics cards unable to report their status

(a) Fault simulation

(i) The microprocessors of the second and third graphics cards were removed in order to simulate failures of the cards and the cards were returned to their slots.

(ii) The console and the system were powered up.

(b) Result

(i) Normal power up commenced.

(ii) The second and third graphics display units did not display test logos.

(iii) The test logo displayed on the first graphics unit was replaced by a fault report indicating that the second and third graphics interface modules failed to report their statuses.

(iv) Advancement to the main demonstration menu was possible.

(c) Result Interpretation

The POST detected the failure of the second and third graphics interface modules to report their statuses.
3.3.2.17 Absence of the first graphics interface module

(a) Fault simulation

(i) The first graphics interface module was removed from the card cage.
(ii) The console and the system were powered up.

(b) Result

(i) Normal power up commenced.
(ii) The first graphics display unit did not display a test logo.
(iii) The test logo displayed on the second graphics display unit was replaced by a fault report indicating that the first graphics interface module was absent.
(iv) Advancement to the main demonstration menu (displayed on the second graphics display unit) was possible.

(c) Result Interpretation

The POST detected the absence of the first graphics interface module card and redirected the error message display to the second graphics display unit via its associated graphics interface module.
3.3.2.18 Absence of the second and third graphics interface modules

(a) Fault simulation

(i) The second and third graphics interface modules were removed.
(ii) The console and the system were powered up.

(b) Result

(i) Normal power up commenced.
(ii) The second and third graphics display unit did not display a test logo.
(iii) The test logo displayed on the first graphics unit was replaced by a fault report indicating that the second and third graphics interface modules were absent.
(iv) Advancement to the main demonstration menu was possible.

(c) Result Interpretation

The POST detected the absence of the second and third graphics interface modules.
3.3.2.19 All graphics cards absent

(a) Fault simulation

(i) All of the graphics interface modules were removed from the system.
(ii) The console and the system were powered up.

(b) Result

(i) Normal power up commenced.
(ii) None of the graphics display units displayed any test logos.
(iii) The RS232 terminal displayed an error message indicating that all of the graphics cards were absent and that the system had been halted. The error code 60 hexadecimal was displayed on the status latch.

(c) Result Interpretation

Having no graphics cards present in the system was defined to be critical since, under normal operational conditions, the graphics displays form an integral part of the man-machine interface. The POST detected the absence of all of the graphics cards and halted the system.
3.3.2.20 Absent system data bus controller card

(a) Fault simulation

(i) The system data bus controller card was removed from its slot.
(ii) The console and the system were powered up.

(b) Result

(i) Normal power up commenced.
(ii) The test logo displayed on the first graphics unit was replaced by a fault report indicating that the system data bus controller card was absent.
(iii) Advancement to the main demonstration menu was possible.

(c) Result Interpretation

The POST detected the absence of the system data bus controller card.
3.3.2.21 Failure of the serial communications to Multibus card to report its status

(a) Fault simulation

(i) The EPROMS were removed from the serial communications to Multibus card and the card returned to its slot.

(ii) The console and the system were powered up.

(b) Result

(i) Normal power up commenced.

(ii) The test logo displayed on the first graphics unit was replaced by a fault report indicating that the serial communications to Multibus failed to report its status.

(iii) Advancement to the main demonstration menu was possible.

(c) Result Interpretation

The POST detected the failure of the serial communications to Multibus card to report its status.
3.3.2.22 Absent serial communications to Multibus card

(a) Fault simulation

(i) The serial communications to Multibus card was removed from its slot.
(ii) The console and the system were powered up.

(b) Result

(i) Normal power up commenced.
(ii) The test logo displayed on the first graphics unit was replaced by a fault report indicating that the serial communications to Multibus card was absent.
(iii) Advancement to the main demonstration menu was possible.

(c) Result Interpretation

The POST detected the absence of the serial communications to Multibus card.
3.3.2.23 Failure of the mass storage controller card to report its status

(a) Fault simulation

(i) The EPROMS were removed from the mass storage controller card and the card returned to its slot.
(ii) The console and the system were powered up.

(b) Result

(i) Normal power up commenced.
(ii) The test logo displayed on the first graphics unit was replaced by a fault report indicating that the mass storage controller card failed to report its status.
(iii) Advancement to the main demonstration menu was possible.

(c) Result Interpretation

The POST detected the failure of the mass storage controller card to report its status.

3.3.2.24 Absent mass storage controller card

(a) Fault simulation

(i) The mass storage controller card was removed from its slot.
(ii) The console and the system were powered up.

(b) Result

(i) Normal power up commenced.
(ii) The test logo displayed on the first graphics unit was replaced by a fault report indicating that the mass storage controller card was absent.
(iii) Advancement to the main demonstration menu was possible.

(c) Result Interpretation

The POST detected the absence of the mass storage controller card.
4 ON-LINE CONSOLE FUNCTIONAL DEMONSTRATIONS

The tests described below were intended to show functionality of the console and its associated on-line fault detection capabilities. All of the demonstrations were performed with the console in its normal operational state and after successful operation of the POST. In all cases the main demonstration menu that appeared on both the first graphics display unit and the RS232 diagnostic terminal looked as shown in Table I.

Table I : Main demonstration menu

<table>
<thead>
<tr>
<th>CONSOLE DEMONSTRATION MAIN MENU</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
<tr>
<td>B</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>D</td>
</tr>
<tr>
<td>E</td>
</tr>
<tr>
<td>F</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Select a letter....

Selection of options A to E resulted in demonstrations run in protected mode. Selection of F resulted in the resetting of the processors and the running of the console [MEHTA, 1990] and mass storage controller card [POLMANS, 1990] demonstrations in real mode. The procedures followed, the results obtained and an interpretation of these results is now given for each of these demonstrations.
4.1 ROLLERBALL MODULE FUNCTIONAL DEMONSTRATION

(a) Demonstration Procedure

(i) The console and the system were powered up.
(ii) The rollerball module demonstration was selected from the main menu.
(iii) The required demonstration was selected from the menu shown in Table II.

<table>
<thead>
<tr>
<th>Table II</th>
<th>Rollerball demonstration menu</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rollerball demonstration menu</td>
<td>A Unmask rollerball</td>
</tr>
</tbody>
</table>

Select a letter.... ESC to quit

(b) Result

(i) The first graphics display unit displayed the $x$ and $y$ coordinates of the cursor and the updating of these values.
(ii) Keypresses were displayed.
(iii) Masking the rollerball module resulted in no response from the rollerball module. It was possible to unmask the rollerball module again.

(c) Result Interpretation

The ability of the primary processor to communicate with the first graphics interface module and control the action of the rollerball module connected to the first graphics interface module was demonstrated.
4.2 GRAPHICS DISPLAY UNIT DEMONSTRATION

(a) Demonstration Procedure

(i) The console and the system were powered up.
(ii) "B" was selected from the main menu in order to invoke the graphics display unit demonstration menu of Table III.
(iii) The required demonstration was selected from the displayed menu.

Table III: Graphics demonstration menus

<table>
<thead>
<tr>
<th>Graphics display unit demonstration menu</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Initialize graphics card</td>
</tr>
<tr>
<td>B</td>
<td>Generate MIDS test pattern</td>
</tr>
<tr>
<td>C</td>
<td>Generate colour bars</td>
</tr>
<tr>
<td>D</td>
<td>Generate circles</td>
</tr>
<tr>
<td>E</td>
<td>Generate softkey pattern</td>
</tr>
<tr>
<td>F</td>
<td>Disable test message generation</td>
</tr>
<tr>
<td>G</td>
<td>Change selection of GDU</td>
</tr>
</tbody>
</table>

Select a letter.... ESC to quit

(b) Result

(i) Selection of "A" resulted in the display of the selected test number and the identification of the current screen below the menu on the RS232 terminal screen. The graphics card driving the currently selected screen was reset and the associated screen displayed the power up logo.
(ii) Selection of "B" resulted in the generation and display of the manual input device test pattern on the current screen.
(iii) Selection of "C" resulted in the generation and display of the colour bar test pattern on the current screen.
(iv) Selection of "D" resulted in the generation and display of the power up test logo on the current screen.
(v) Selection of "E" resulted in the generation and display of the softkey pattern on the current screen.
(vi) Selection of "F" resulted in the disabling of the graphics display unit echo of
the RS232 display.

(vii) Selection of "G" resulted in the display of the menu shown in Table IV which
enabled the selection of an alternative screen. The above steps were then
repeated for the different screens.

(viii) When the graphics card associated with a specific screen was absent from
the system, it was reported that it was not possible to select the pertinent
screen.

<table>
<thead>
<tr>
<th>Table IV</th>
<th>: Graphics display unit selection menu</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graphics display unit selection menu</td>
<td></td>
</tr>
<tr>
<td>A Select GDU 1</td>
<td></td>
</tr>
<tr>
<td>B Select GDU 2</td>
<td></td>
</tr>
<tr>
<td>C Select GDU 3</td>
<td></td>
</tr>
</tbody>
</table>

Select a letter...ESC to quit

(c) Result interpretation

The ability of the primary processor to communicate with and detect the absence of
the graphics interface modules was demonstrated.
4.3 MULTIMASTER BUS CONTENTION DEMONSTRATION

(a) Demonstration Procedure

(i) The main applications processor card cage was withdrawn and the RS232 terminal connected to the serial port of the secondary processor.

(ii) The interlock override was pressed and the console and system were powered up.

(iii) The multimaster capabilities demonstration was selected from the main menu.

(b) Result

(i) The number of transitions recorded by the primary processor were displayed on the first graphics display unit and that of the secondary processor on the RS232 terminal. These numbers were positive and equal.

(c) Result Interpretation

A global variable was alternately set and reset by the primary and secondary processors, respectively. The ability of the system to allow more than one master to control the bus was demonstrated. In this case, three bus masters shared the bus, namely the two CPU cards and the first graphics interface module.
4.4 LOCAL AREA NETWORK FUNCTIONAL DEMONSTRATION

4.4.1 Normal operation

(a) Demonstration Procedure

(i) The console and the system were powered up.

(ii) "E" was selected from the main menu in order to display the serial communications to Multibus demonstration menu shown in Table V.

(iii) The required option was selected from this menu.

(b) Result

(i) Selecting "A" resulted in the board performing a self-test and displaying the result.

(ii) Selecting "B" reset the serial communications to Multibus card. This was signified by the extinguishing of the run LED on the front panel of the card for the duration of the reset pulse.

(iii) Selection of "C" allowed the selection of a number of link tests as indicated in the menu of Table VI. For each of the first four options the board transmitted a buffer over the selected link at the current baud rate.

(iv) Selection of "D" enabled the baud rate to be altered via selections from the menu of Table VII.

Table V: Serial communications to Multibus menu

<table>
<thead>
<tr>
<th>Serial communications to Multibus demonstration menu</th>
</tr>
</thead>
<tbody>
<tr>
<td>A Perform board self-test</td>
</tr>
<tr>
<td>B Reset serial communications to Multibus card</td>
</tr>
<tr>
<td>C Invoke HDLC menu</td>
</tr>
</tbody>
</table>

Select a letter... ESC to quit
(c) Result Interpretation

The ability to test and reset the serial communications to Multibus card was demonstrated as well as the ability to transmit data at varying speeds.
### Table VI: High-level data link control menu

**HDLC menu**

<table>
<thead>
<tr>
<th>Rx chan</th>
<th>Tx chan</th>
<th>Calculated task speed</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>A Test 1</td>
<td>1A</td>
<td>2A</td>
<td>BAUD</td>
</tr>
<tr>
<td>B Test 2</td>
<td>1B</td>
<td>2B</td>
<td>BAUD</td>
</tr>
<tr>
<td>C Test 3</td>
<td>2A</td>
<td>1A</td>
<td>BAUD</td>
</tr>
<tr>
<td>D Test 4</td>
<td>2B</td>
<td>1B</td>
<td>BAUD</td>
</tr>
<tr>
<td>E Select HDLC link speed.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Current HDLC link setting: XX Kb/s

Select a letter.... ESC to quit

### Table VII: HDLC baud rate selection menu

**Speed selection menu**

Current setting: 15 Kb/s

| A 15 | B 20 |
| C 25 | D 30 |
| E 35 | F 40 |
| G 45 | H 50 |
| I 55 | J 60 |
| K 65 | L 70 |
| M 75 | N 80 |
| O 90 | P 100 |

Select a letter.... ESC to quit
4.4.2 Local area network fault detection

(a) Demonstration Procedure

(i) The console and the system were powered up.

(ii) "E" was selected from the main menu, resulting in the serial communications to Multibus menu of Table V.

(iii) "C" was selected in order to display the HDLC demonstration menu of Table VI.

(iv) The loopback plug was removed from socket SK9 on the back of the feedthrough module.

(v) The required option was selected ("A" to "E").

(vi) The loopback plug was re-inserted into the socket.

(b) Result

(i) Each time one of the first four options were invoked, the terminal reported the failure of the board to transmit the data.

(ii) Once the plug was reinstated, the test that failed previously, passed.

(c) Result Interpretation

The transmission of the data via the feedback plug was demonstrated. This plug would be used to validate the operation of the serial communications to Multibus in the event of a failure on the local area network.
4.5 SYSTEM RESET DEMONSTRATION

(a) Demonstration Procedure

(i) The console and the system were powered up.
(ii) "F" was selected from the main menu in order to reset the system and run the real mode demonstrations.

(b) Result

(i) On selecting "F", the system reset. This was indicated by the green LED on the bus terminator unit illuminating for the duration of the reset pulse.
(ii) The two CPU cards reset and performed a "warm" start. The primary CPU card displayed the real mode demonstration menu of Table VIII and the secondary processor reported its recognition of a "warm" start and subsequently halted execution.

<table>
<thead>
<tr>
<th>Table VIII</th>
<th>: Real mode demonstration menu</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real mode demonstration menu</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>Perform console demonstration</td>
</tr>
<tr>
<td>B</td>
<td>Perform mass storage unit demonstration</td>
</tr>
<tr>
<td>Select a letter....</td>
<td>ESC to quit</td>
</tr>
</tbody>
</table>

(c) Result Interpretation

The ability of the system to execute a software controlled reset was demonstrated. The secondary processor was halted, since it was not needed to perform the real mode demonstrations.
(a) Demonstration Procedure

(i) The console and the system were powered up.

(ii) "F" was selected from the main menu, resulting in the real mode demonstration menu of Table VIII.

(iii) "A" was chosen from the real mode menu to invoke the console demonstration menu shown in Table IX. This allowed the selection of demonstrations of the applications processor interface functions and the manual input devices of the console.

(iv) "A" was selected from this menu to invoke the applications processor interface demonstration menu illustrated in Table X.

(v) The required demonstration was then selected from the displayed menu.

Table IX

: Console demonstration menu

| A | Applications processor interface demonstration |
| B | Manual input devices demonstration |

Select a letter.... ESC to quit

Table X

: Applications processor interface demonstration menu

| A | Reset API and all MIDS |
| B | Invoke API self-test |
| C | Perform audible alarm test |

Select a letter.... ESC to quit
(b) Result

(i) Selection of "A" from the applications processor interface demonstration menu of Table X resulted in the resetting of the applications processor interface and all the manual input devices together with a message indicating that the test had passed.

(ii) Selection of "B" from the applications processor interface demonstration menu of Table X resulted in the applications processor interface invoking a self-test and testing its communication ports. A message indicating that the tests had passed was also displayed on the RS232 terminal.

(iii) Selection of "C" from the applications processor interface demonstration menu of Table X resulted in a prompt for the number of beeps to be sounded by the audible alarm. Once this number was entered, the alarm sounded the required number of times.

(c) Result Interpretation

The ability of the primary processor to communicate with the applications processor interface card and determine the status of devices connected to the latter was demonstrated.

4.7 MANUAL INPUT DEVICES FUNCTIONAL DEMONSTRATION

(a) Demonstration Procedure

(i) The console and the system were powered up.

(ii) "F" was selected from the main menu, resulting in the real mode demonstration menu of Table VIII.

(iii) "A" was selected from the real mode demonstration menu, resulting in the console demonstration menu of Table IX.

(iv) "B" was selected in order to invoke the manual input device menu illustrated in Table XI.

(v) The required demonstration was selected from the displayed menu.
Table XI: Manual input devices demonstration menu

<table>
<thead>
<tr>
<th>Manual input devices demonstration menu</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
<tr>
<td>B</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>D</td>
</tr>
<tr>
<td>E</td>
</tr>
<tr>
<td>F</td>
</tr>
</tbody>
</table>

Select a letter... ESC to quit

(b) Result

(i) Selection of "A", "B" or "C" from the manual input devices demonstration menu resulted in the display of the softkey module demonstration menu which allowed the selection of the listed demonstrations shown in Table XII for each softkey module.

(ii) Selection of option "D" from the manual input devices demonstration menu resulted in the display of the "qwerty" keyboard demonstration menu which allowed the selection of the demonstrations as illustrated in Table XIII.

(v) Selection of "E" from the manual input devices demonstration menu resulted in the display of the environmental monitoring and control demonstration menu which allowed the selection of the demonstrations listed in Table XIV.

(vi) Selection of "F" from the manual input devices demonstration menu resulted in the display of the status of each of the input devices (e.g. the environmental monitoring and control card, the three softkey modules and the "qwerty" keyboard were displayed as "connected"). Pressing any of the keys of the four manual input devices resulted in the display of a message identifying the module and the key pressed.

(c) Result Interpretation

The ability of the primary processor to communicate with the applications processor interface card and determine the status of devices connected to the latter was demonstrated.
### Table XII: Softkey module demonstration menu

Softkey module demonstration menu

A  | Reset SKM
B  | Mask SKM
C  | Unmask SKM
D  | Enable module stuck key detect
E  | Disable module stuck key detect
F  | Mask a key
G  | Unmask a key
H  | Enable stuck key detect
I  | Disable stuck key detect

Select a letter....  ESC to quit

### Table XIII: "Qwerty" keyboard demonstration menu

Qwerty keyboard module demonstration menu

A  | Reset QKBM
B  | Mask QKBM
C  | Unmask QKBM
D  | Enable module stuck key detect
E  | Disable module stuck key detect
F  | Mask a bank
G  | Unmask a bank
H  | Enable bank stuck key detect
I  | Disable bank stuck key detect

Select a letter....  ESC to quit

### Table XIV: Environmental monitoring and control menu

Environmental monitoring and control demonstration menu

A  | Status of console
B  | Status request menu
C  | Control mode menu
D  | Reset EMAC

Select a letter....  ESC to quit
4.7.1 Softkey modules functional demonstration

(a) Demonstration Procedure

(i) The console and the system were powered up.
(ii) "F" was selected from the main menu, resulting in the real mode demonstration menu of Table VIII.
(iii) "A" was selected from the real mode demonstration menu, resulting in the console demonstration menu of Table IX.
(iv) "B" was selected in order to invoke the manual input device menu illustrated in Table XI.
(v) The required softkey module was chosen in order to demonstrate their functionality.

(b) Result

For each of the softkey modules the following resulted:

(i) Masking a module resulted in no response from keys on the masked module. It was possible to unmask the module again.
(ii) Enabling module stuck key detect resulted in the detection and automatic masking of the stuck key. Clearing the module stuck key detect resulted in notification of the clearance occurring.
(iii) Masking a key resulted in no response from the masked key. Adjacent keys were not affected. It was possible to unmask the keys again.
(iv) Enabling stuck key detect resulted in the detection and automatic masking of that stuck key. Disabling the stuck key detect resulted in notification of the disabling occurring. Adjacent keys were not affected and were capable of autorepeat if the applicable stuck key detect facility was disabled.

(c) Result Interpretation

The ability of the primary processor to communicate with the applications processor interface and determine the status of the softkey module devices connected to the applications processor interface was demonstrated.
4.7.2 "Qwerty" keyboard module functional demonstration

(a) Demonstration Procedure

(i) The console and the system were powered up.

(ii) "F" was selected from the main menu, resulting in the real mode demonstration menu of Table VIII.

(iii) "A" was selected from the real mode demonstration menu, resulting in the console demonstration menu of Table IX.

(iv) "B" was selected in order to invoke the manual input device menu illustrated in Table XI.

(v) The "qwerty" keyboard module menu was invoked in order to demonstrate its functionality.

(b) Result

(i) Masking the "qwerty" keyboard module resulted in no response from keys on the "qwerty" keyboard module. It was not possible to unmask the "qwerty" keyboard module once it had been masked without a system reset.

(ii) Enabling stuck key detect resulted in the detection of the stuck key and automatic masking of the affected bank. Clearing the stuck key detect facility resulted in notification of the clearance occurring.

(iii) Masking a bank resulted in no response from the keys on that bank. Adjacent banks were not affected. It was possible to unmask these banks again.

(iv) Enabling the bank stuck key detect facility resulted in the detection of the stuck key and the automatic masking of the bank. Once again, disabling the stuck key detect facility resulted in notification of such disabling. Adjacent banks were not affected and were capable of autorepeat, provided that the associated bank stuck key detect facility was disabled.

(c) Result Interpretation

The ability of the primary processor to communicate with the applications processor interface and determine the status of "qwerty" keyboard module connected to the applications processor interface was demonstrated.
4.8 ENVIRONMENTAL MONITORING AND CONTROL FUNCTIONAL DEMONSTRATION

4.8.1 Environmental monitoring and control normal operation

4.8.1.1 Lamp brightness/test switch

(a) Demonstration Procedure

(i) The console and the system were powered up.
(ii) The lamp test button on the status and control panel was pressed.

(b) Result

(i) All control and status panel module lamps and LEDs illuminated.
(ii) Holding the switch down resulted in the intensity of the lamps continuously increasing until a maximum brightness was attained and then decreasing until a minimum brightness was attained.

(c) Result Interpretation

The ability to test and control the brightness of the lamps was demonstrated. Correct functioning of the LEDs was also confirmed.
4.8.1.2 Console Status Display

(a) Demonstration Procedure

(i) The console and the system were powered up.
(ii) "F" was selected from the main menu, resulting in the real mode demonstration menu of Table VIII.
(iii) "A" was selected from the real mode demonstration menu, resulting in the console demonstration menu of Table IX.
(iv) "B" was selected in order to invoke the manual input device menu illustrated in Table XI.
(v) "E" was selected from the manual input devices menu in order to display the environmental monitoring and control menu of Table XIV.
(iv) "A" was then chosen from this menu to display the status of the entire console.
(v) Steps (i) to (iii) were repeated and "B" was selected to display the menu of status requests shown in Table XV.

Table XV: Console status request menu

<table>
<thead>
<tr>
<th>Status request menu</th>
</tr>
</thead>
<tbody>
<tr>
<td>A   Temperature</td>
</tr>
<tr>
<td>B   Humidity</td>
</tr>
<tr>
<td>C   EMAC operating mode</td>
</tr>
<tr>
<td>D   EMAC self-test</td>
</tr>
<tr>
<td>E   Anti-condensation heating</td>
</tr>
<tr>
<td>F   Power supply menu</td>
</tr>
<tr>
<td>G   Interlocks menu</td>
</tr>
<tr>
<td>H   Control and status panel menu</td>
</tr>
<tr>
<td>I   Ventilation voltage</td>
</tr>
</tbody>
</table>

Select a letter... ESC to quit

(b) Result

(i) When "A" was selected from the environmental monitoring and control menu of Table XIV, console information pertaining to parameters monitored and controlled by the environmental monitoring and control were displayed on the test terminal.
(ii) When the status request menu of Table XV was invoked, the test terminal displayed the current status of the selected parameter for each status request that was made. For example, if a temperature request was issued, the test terminal responded with \textsc{temperature} = \textless \textit{data} \textgreater \textsc{kelvin}. For certain parameter status displays (options "F", "G" and "I" of Table XV) more menus were displayed as illustrated in Table XVI and further operator input was required.

(c) Result Interpretation

The ability to determine the current status of individual console parameters was demonstrated.

\begin{table}[h]
\centering
\begin{tabular}{l}
\textbf{Table XVI} : Further status request menus \\

\hline
\multicolumn{1}{c}{Power supply menu} \\
\hline
A & Power supply unit 1 5V \\
B & Power supply unit 1 12V \\
C & Mains \\
D & Frequency \\
E & Auxiliaries \\
F & Power supply unit 2 5V \\
G & Power supply unit 2 12V \\
\hline
Select a letter.... & ESC to quit \\
\end{tabular}
\end{table}

\begin{table}[h]
\centering
\begin{tabular}{l}
\multicolumn{1}{c}{Interlock menu} \\
\hline
A & Graphics display unit 1 \\
B & Graphics display unit 2 \\
C & Graphics display unit 3 \\
D & Card cage 1 \\
E & Card cage 2 \\
F & Fan housing/drawer \\
\hline
Select a letter.... & ESC to quit \\
\end{tabular}
\end{table}

\begin{table}[h]
\centering
\begin{tabular}{l}
\multicolumn{1}{c}{Control and status panel module switch menu} \\
\hline
A & Bank 1 \\
B & Bank 2 \\
\hline
Select a letter.... & ESC to quit \\
\end{tabular}
\end{table}
4.8.1.3 User Defined Lamps and LEDs functional demonstration

(a) Demonstration Procedure

(i) The console and the system were powered up.

(ii) "F" was selected from the main menu, resulting in the real mode demonstration menu of Table VIII.

(iii) "A" was selected from the real mode demonstration menu, resulting in the console demonstration menu of Table IX.

(iv) "B" was selected in order to invoke the manual input device menu illustrated in Table XI.

(v) "E" was selected from the manual input devices menu in order to display the environmental monitoring and control menu of Table XIV.

(vi) "C" was selected in order to display the control mode menu of Table XVII.

(vii) Options "A" to "D" were then selected.

Table XVII: Control mode menu

<table>
<thead>
<tr>
<th>Control mode menu</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
<tr>
<td>B</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>D</td>
</tr>
</tbody>
</table>

Select a letter... ESC to quit
(b) Result

(i) Selecting option "A" or "B" resulted in all the user defined lamps being illuminated or extinguished.

(ii) Choosing options "C" or "D" resulted in all the user defined LEDs being illuminated or extinguished.

(c) Result Interpretation

The ability to control the user defined lamps and LEDs was demonstrated.

4.8.2 Environmental monitoring and control fault detection

(a) Demonstration Procedure

(i) The console and the system were powered up.

(ii) "F" was selected from the main menu, resulting in the real mode demonstration menu of Table VIII.

(iii) "A" was selected from the real mode demonstration menu, resulting in the console demonstration menu of Table IX.

(iv) "B" was selected in order to invoke the manual input device menu illustrated in Table XI.

(v) The console functionality test (option "F") was then selected from this menu.

(vi) The first graphics display unit drawer was withdrawn and closed again.

(vii) The first softkey module was disconnected and then reconnected.

(viii) The temperature sensor was immersed in a cup of cold water.

(ix) After sufficient time had elapsed to record the results, a heat gun was directed at the temperature sensor.

(x) After a long enough delay, the heat source was removed.

(b) Result

(i) When the first graphics unit drawer was withdrawn, the power to the unit was switched off and the INT LED and WNG status lamp were switched on. The seven segment code on the environmental monitoring and control card displayed a three and the test terminal displayed that the first graphics display unit drawer was open.
(ii) On closing the drawer, power to the first graphics display unit was restored. The INT LED and WNG lamp switched off and the seven segment code on the environmental monitoring and control card was reset to zero. The status of the drawer displayed on the terminal changed back to reflect that the drawer was closed.

(iii) On disconnecting the softkey module, the WNG lamp and MIDS LED switched on. The seven segment displayed a four and the terminal displayed a fault detected on the first softkey module.

(iv) As soon as the softkey module was reconnected, the WNG lamp and MIDS LED were switched off again. The seven segment display was reset and the test terminal displayed the reconnected status.

(v) When the temperature reading displayed on the test terminal dropped to 283 Kelvin, the fan speed of the processor card cage was reduced to a slow speed. This was monitored both visually and audibly.

(vi) On the other hand, when the temperature reading displayed on the test terminal increased to above 313 Kelvin, the TEMP LED was switched on. The seven segment display on the environmental monitoring and control card showed code A hexadecimal and the fan speed increased to its maximum capability. When the Temperature reading reached above 323 Kelvin, the seven segment display showed code B hexadecimal and power was shut down to all the controlled units.

(vii) On removing the heat source, the temperature of the sensor fell slowly back to room temperature. When the seven segment display showed an A hexadecimal, power was restored to all elements. On reaching normal operating temperature, the display was reset to zero and the TEMP LED was switched off.

(c) Result Interpretation

This demonstration showed the ability of the environmental monitoring and control to monitor parameters, interpret this monitoring, and route the appropriate information via the applications processor interface to the primary processor.
4.9 SYSTEM OVERRIDE FUNCTIONAL DEMONSTRATION

(a) Demonstration Procedure

(i) The console and the system were powered up.
(ii) "F" was selected from the main menu, resulting in the real mode demonstration menu of Table VIII.
(iii) "A" was selected from the real mode demonstration menu, resulting in the console demonstration menu of Table IX.
(iv) "B" was selected in order to invoke the manual input device menu illustrated in Table XI.
(v) The console functionality test (option "F") was selected from this menu.
(vi) A heat gun was directed at the temperature sensor.
(vii) When the environmental monitoring and control card had shut the system down, the system override switch was pressed and then released.

(b) Result

(i) The system powered down on reaching the threshold temperature.
(ii) When the system override switch was pressed, its associated status indication switched on and power was applied to the system.
(iii) When the system override switch was released, the system powered down again.

(c) Result Interpretation

The ability of the system override switch to bypass the control of the environmental monitoring and control was demonstrated.
4.10 INTERLOCK OVERRIDE FUNCTIONAL DEMONSTRATION

(a) Demonstration Procedure

(i) The console and the system were powered up.
(ii) The interlock override switch was pressed.
(iii) One of the drawers was withdrawn.
(iv) The interlock override switch was released.
(v) The procedure was repeated for each of the console drawers.

(b) Result

(i) When the interlock override switch was operated its status indication switched on.
(ii) After withdrawing each of the drawers, the power applied to the drawer remained on.
(iii) Power on a withdrawn drawer was removed when the switch was released.

(c) Result Interpretation

The ability of the interlock override switch to bypass the interlocks was demonstrated. This switch allowed for the withdrawal of the main applications processor card cage to observe the error codes appearing on the bus terminator unit.
4.11 MASS STORAGE CONTROLLER DEMONSTRATION

In all of the demonstrations of the mass storage controller card, the console and system were powered up and option "F" was selected from the main demonstration menu. As was mentioned, this resulted in the resetting of the processors and the resultant real mode demonstration menu of Table VIII. Selection of option "B" from this menu resulted in the display of the mass storage controller card demonstration menu of Table XVIII.

<table>
<thead>
<tr>
<th>Table XVIII</th>
<th>Mass storage controller card demonstration menu</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mass storage demonstration menu</td>
</tr>
<tr>
<td>A</td>
<td>Format floppy disk</td>
</tr>
<tr>
<td>B</td>
<td>Read directory</td>
</tr>
<tr>
<td>C</td>
<td>Read a file</td>
</tr>
<tr>
<td>D</td>
<td>Copy a file</td>
</tr>
<tr>
<td>E</td>
<td>Delete a file</td>
</tr>
<tr>
<td>F</td>
<td>Write a new file</td>
</tr>
<tr>
<td>G</td>
<td>Fault simulation menu</td>
</tr>
</tbody>
</table>

Select a letter.... ESC to quit
4.11.1 Mass storage controller normal operation

4.11.1.1 Formatting a floppy disk

(a) Demonstration Procedure

(i) A floppy disk was inserted into the disk drive of the console.
(ii) The procedure to format a floppy disk was invoked.
(iii) The format was verified by inserting the floppy disk into the disk drive of a personal computer.

(b) Result

(i) Visual inspection of the disk drive showed the heads stepping in towards the centre while the disk was formatting.
(ii) On completion, a message indicating that no errors had occurred appeared on the test terminal.
(iii) When the disk was inserted into the disk drive of the personal computer, it was found to be formatted correctly and contain the label of "TEST FORMAT" and no files.

(c) Result Interpretation

The processor was able to instruct the mass storage controller card to format a 360K disk.

---

1 The floppy disk and personal computer referred to were both IBM (360K) compatible.
4.11.1.2 Reading a directory

(a) Demonstration Procedure

(i) A formatted floppy disk with a known directory listing of a number of ASCII files was inserted into the floppy disk drive.

(iv) The procedure to read the directory was invoked.

(b) Result

(i) A directory listing of the contents of the disk was displayed on the screen.

(ii) On completion, a message indicating that no errors had occurred appeared on the test terminal.

(iii) The directory listing corresponded to the list obtained from the same floppy disk on the personal computer.

(c) Result Interpretation

The mass storage controller card was able to read the root directory of the formatted floppy disk, and pass the contents back to the processor via the dual port ram on the mass storage controller card.

4.11.1.3 Reading a file

(a) Demonstration Procedure

(i) A formatted floppy disk with a number of ASCII files was inserted into the floppy disk drive.

(ii) The procedure to read a file was invoked and a known filename was entered when the system requested a filename.
(b) Result

(i) The contents of the named file together with a message indicating that no errors had occurred appeared on the test terminal.

(c) Result Interpretation

The mass storage controller card was capable of reading and interpreting the directory information and the file allocation tables on the floppy disk. Furthermore, it was able to pass the contents of the disk file back to the primary processor using the dual port ram on the mass storage controller card.

4.11.1.4 Copying a file

(a) Demonstration Procedure

(i) A formatted floppy disk with a number of ASCII files was inserted into the floppy disk drive.

(ii) The procedure to copy a file was invoked and a known filename was entered when the system requested a source filename. A file not existing on the disk was given for the destination file.

(iii) The copy was verified on the personal computer.

(b) Result

(i) The file was copied correctly and a message indicating that no errors had occurred appeared on the test terminal.

(c) Result Interpretation

The mass storage controller card was able to copy a source file to a destination file without any errors occurring.
4.11.1.5 Deleting a file

(a) Demonstration Procedure

(i) A formatted floppy disk with a number of ASCII files was inserted into the floppy disk drive.
(ii) The procedure to delete a file was invoked and a known existing filename was entered when the system requested the filename.
(iii) The deletion was verified on the personal computer.

(b) Result

(i) When the system had deleted the file, a message indicating that no errors had occurred appeared on the test terminal.
(ii) The directory showed that the file was erased by substituting a "\sigma" into the position of the first character of the filename.

(c) Result Interpretation

The mass storage controller card was capable of deleting files on the floppy disk.

4.11.1.6 Writing a file

(a) Demonstration Procedure

(i) A formatted floppy disk was inserted into the floppy disk drive.
(ii) The procedure to write a new file was invoked.
(iii) The contents of the new file were read by invoking the procedure to read a file and were also verified on the personal computer.
(b) Result

(i) A new file ALPHA.BET was created on the disk and a message indicating that no errors had occurred appeared on the test terminal.

(ii) Reading the file resulted in the following display on the terminal:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz)!@#$%&*()
```

(c) Result Interpretation

The mass storage controller card was able to write the contents of a block of global memory to a floppy disk, and adjust the root directory and file allocation tables to reflect the creation of this new file.

4.11.2 Mass storage controller fault detection

4.11.2.1 Reading a non-existent file

(i) A formatted floppy disk with a number of ASCII files was inserted into the floppy disk drive.

(ii) The procedure to read a file was invoked and a non-existent filename was entered when the system requested a filename.

(b) Result

(i) A message appeared on the test terminal indicating that the file could not be found.

(c) Result Interpretation

This test demonstrated the ability of the mass storage controller card to determine that an attempt was made to read a non-existent file.
4.11.2.2 Writing a file that already existed

(i) A formatted floppy disk with a number of ASCII files was inserted into the floppy disk drive.

(ii) An attempt was made to write an existing file to the floppy disk.

(b) Result

(i) A message appeared on the test terminal indicating that the file already existed and the option was provided to replace the existing file.

(c) Result Interpretation

This test demonstrated the ability of the mass storage controller card to determine that an attempt was made to write a file that already existed.

4.11.2.3 Code simulated faults

(a) Demonstration procedure

(i) The fault simulation menu of Table XIX was invoked by choosing option "G" of the mass storage controller card demonstration menu of Table XVIII.

(ii) The appropriate fault simulation was chosen from this menu.

Table XIX: Mass storage controller card fault simulation menu

<table>
<thead>
<tr>
<th>Mass storage fault simulation menu</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
<tr>
<td>B</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>D</td>
</tr>
<tr>
<td>E</td>
</tr>
<tr>
<td>F</td>
</tr>
<tr>
<td>G</td>
</tr>
<tr>
<td>H</td>
</tr>
</tbody>
</table>

Select a letter... ESC to quit
(b) Result

(i) Writing a file from ROM that was larger than 64K, resulted in the file being truncated and an appropriate error message to this effect.

(ii) For each invalid parameter that was passed to the mass storage controller card, the card responded with a relevant message.

(c) Result interpretation

The ability of the mass storage controller card to realise certain erroneous conditions was demonstrated by this test procedure.
5 OFF-LINE DIAGNOSTICS

Selection of option "D" from the main demonstration menu of Table I resulted in the display of the menu presented in Table XX. In all of the off-line diagnostics tests listed below, the system was powered up and option "D" was chosen. The individual tests were then each conducted in turn. Any tests that were attempted on a non-critical card that was absent from the system resulted in a report to the RS232 terminal informing the operator that the card was missing.

Table XX: Primary processor off-line diagnostics menu

<table>
<thead>
<tr>
<th>Letter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Primary processor self-test</td>
</tr>
<tr>
<td>B</td>
<td>Primary processor EPROM/RAM test</td>
</tr>
<tr>
<td>C</td>
<td>Primary processor dynamic RAM test</td>
</tr>
<tr>
<td>D</td>
<td>First graphics interface module test</td>
</tr>
<tr>
<td>E</td>
<td>Second graphics interface module test</td>
</tr>
<tr>
<td>F</td>
<td>Third graphics interface module test</td>
</tr>
<tr>
<td>G</td>
<td>Serial communications to Multibus test</td>
</tr>
<tr>
<td>H</td>
<td>Applications processor interface test</td>
</tr>
<tr>
<td>I</td>
<td>System data bus controller test</td>
</tr>
<tr>
<td>J</td>
<td>Mass storage controller test</td>
</tr>
<tr>
<td>K</td>
<td>Secondary processor test</td>
</tr>
</tbody>
</table>

Select a letter...ESC to quit
5.1 PRIMARY PROCESSOR TESTS

5.1.1 Primary processor self-test

(a) Demonstration Procedure

(i) The console and the system were powered up.
(ii) "D" and "A" were selected in order to invoke the primary processor self-test.

(b) Result

(i) The primary processor conducted a self-test and reported the result on the RS232 terminal.

(c) Result Interpretation

The ability to invoke a self-test on the primary processor was demonstrated.

5.1.2 Primary processor EPROM/RAM card test

(a) Demonstration Procedure

(i) The console and the system were powered up.
(ii) "D" and "B" were selected in order to invoke the primary processor EPROM/RAM test.

(b) Result

(i) The primary processor conducted a checksum test on the ROM of the EPROM/RAM card and tested the RAM. The result was returned to the RS232 terminal.

(c) Result Interpretation

The ability of the primary processor to test the EPROM/RAM card was demonstrated.
5.1.3 Primary processor dynamic RAM card test

(a) Demonstration Procedure

(i) The console and the system were powered up.
(ii) "D" and "C" were selected in order to invoke the primary processor dynamic RAM test.

(b) Result

(i) The primary processor tested the dynamic RAM card via both busses and returned the appropriate result.

(c) Result Interpretation

The ability of the primary processor to test the dynamic RAM was demonstrated.
5.1.4 Primary processor graphics interface modules test

(a) Demonstration Procedure

(i) The console and the system were powered up.
(ii) "D" and "D" were selected in order to invoke the first graphics interface module self-test.
(iii) The procedure was repeated for the second and third graphics interface modules by selecting "E" and "F" from the primary processor off-line diagnostics menu, respectively.

(b) Result

(i) For each of the above tests, the primary processor invoked a self-test on the appropriate graphics interface module and returned the result.

(c) Result Interpretation

The ability of primary processor to obtain the status of the graphics cards was demonstrated.
5.1.5 Primary processor serial communications to Multibus card test

(a) Demonstration Procedure

(i) The console and the system were powered up.

(ii) "D" and "G" were selected in order to invoke the serial communications to Multibus self-test.

(b) Result

(i) The primary processor invoked a self-test on the serial communications to Multibus card and returned the result to the RS232 terminal.

(c) Result Interpretation

The ability of the primary processor to test the serial communications to Multibus was demonstrated.

5.1.6 Primary processor applications processor interface card test

(a) Demonstration Procedure

(i) The console and the system were powered up.

(ii) "D" and "H" were selected in order to invoke the applications processor interface test.

(b) Result

(i) The primary processor returned the result of invoking a self-test and testing the dual port RAM of the applications processor interface card to the RS232 terminal.

(c) Result Interpretation

The ability of primary processor to test the applications processor interface card was demonstrated.
5.1.7 Primary processor system data bus controller card test

(a) Demonstration Procedure

(i) The console and the system were powered up.
(ii) "D" and "I" were selected in order to invoke the system data bus controller card test.

(b) Result

(i) The primary processor invoked a self-test and tested the dual port ram of the system data bus controller card. The result of the test was displayed on the RS232 terminal.

(c) Result Interpretation

The ability of primary processor to test the system data bus controller card was demonstrated.
5.1.8 Primary processor mass storage controller card test

(a) Demonstration Procedure

(i) The console and the system were powered up.

(ii) "D" and "J" were selected in order to invoke the mass storage controller card test.

(b) Result

(i) The primary processor invoked a self-test and tested the dual port ram of the mass storage controller card. The result of the test was displayed on the RS232 terminal.

(c) Result Interpretation

The ability of primary processor to test the mass storage controller card was demonstrated.
5.1.9 Primary processor test on the secondary processor

(a) Demonstration Procedure

(i) The console and the system were powered up.
(ii) "D" and "K" were selected in order to invoke the secondary processor test.

(b) Result

(i) The primary processor test menu was replaced by the secondary processor off-line diagnostic menu as shown in Table XXI. In addition to allowing for the testing of the secondary processor, selecting an option from this menu also enabled the testing of the other cards by the secondary processor.

Table XXI: Secondary processor off-line diagnostic menu

<table>
<thead>
<tr>
<th>Secondary processor off-line diagnostics menu</th>
</tr>
</thead>
<tbody>
<tr>
<td>A Secondary processor self test</td>
</tr>
<tr>
<td>B Secondary processor eprom/ram test</td>
</tr>
<tr>
<td>C Secondary processor dynamic RAM test (Multibus only)</td>
</tr>
<tr>
<td>D First graphics interface module test</td>
</tr>
<tr>
<td>E Second graphics interface module test</td>
</tr>
<tr>
<td>F Third graphics interface module test</td>
</tr>
<tr>
<td>G Serial communications to Multibus test</td>
</tr>
<tr>
<td>H Applications processor interface test</td>
</tr>
<tr>
<td>I System data bus controller test</td>
</tr>
<tr>
<td>J Mass storage controller test</td>
</tr>
</tbody>
</table>

Select a letter.... ESC to quit

(c) Result Interpretation

The ability to pass testing requirements to the secondary processor was demonstrated.
5.2 SECONDARY PROCESSOR TESTS

5.2.1 Secondary processor self-test

(a) Demonstration Procedure

(i) The console and the system were powered up.
(ii) "D" and "K" were selected in order to invoke the secondary processor test.
(iii) "A" was selected in order to invoke the secondary processor self-test.

(b) Result

(i) The secondary processor executed a self-test and returned the result to the RS232 terminal.

(c) Result Interpretation

The ability of primary processor to invoke a self-test of on the secondary processor was demonstrated.
5.2.2 Secondary processor EPROM/RAM test

(a) Demonstration Procedure

(i) The console and the system were powered up.
(ii) "D" and "K" were selected in order to invoke the secondary processor test.
(iii) "B" was selected in order to invoke the secondary processor EPROM/RAM test.

(b) Result

(i) The EPROM/RAM card associated with the secondary processor was tested by the latter and the result was displayed on the RS232 terminal.

(c) Result Interpretation

The ability of the secondary processor to test its associated EPROM/RAM card was demonstrated.
5.2.3 Secondary processor dynamic RAM test

(a) Demonstration Procedure

(i) The console and the system were powered up.
(ii) "D" and "K" were selected in order to invoke the secondary processor test.
(iii) "C" was selected in order to invoke the secondary processor dynamic RAM test.

(b) Result

(i) The secondary processor tested the Multibus access to the dynamic RAM card and reported the result to the RS232 terminal.

(c) Result Interpretation

The ability for the secondary processor to test the dynamic RAM card was demonstrated.
5.2.4 Secondary processor graphics interface modules test

(a) Demonstration Procedure

(i) The console and the system were powered up.
(ii) "D" and "K" were selected in order to invoke the secondary processor test.
(iii) "D" was selected in order to invoke the first graphics interface module test.
(iv) The test was repeated for second and third graphics interface modules by selecting "E" and "F" of the secondary processor off-line diagnostics menu, respectively.

(b) Result

(i) For each of the above selections, the secondary processor invoked a self-test of the appropriate graphics card and displayed the result of the test on the RS232 terminal.

(c) Result Interpretation

The ability of the secondary processor to test the graphics cards was demonstrated.
5.2.5 The secondary processor serial communications to Multibus card test

(a) Demonstration Procedure

(i) The console and the system were powered up.
(ii) "D" and "K" were selected in order to invoke the secondary processor test.
(iii) "G" was selected in order to invoke the serial communications to Multibus card self-test.

(b) Result

(i) The secondary processor invoked the serial communications to Multibus card self-test and displayed the result on the RS232 terminal.

(c) Result Interpretation

The ability of secondary processor to test the serial communications to Multibus card was demonstrated
5.2.6 Secondary processor applications processor interface card test

(a) Demonstration Procedure

(i) The console and the system were powered up.
(ii) "D" and "K" were selected in order to invoke the secondary processor test.
(iii) "H" was selected in order to invoke the applications processor interface card test.

(b) Result

(i) The secondary processor tested the dual port RAM of the applications processor interface, invoked an applications processor interface self-test, and returned the result to the RS232 terminal.

(c) Result Interpretation

The ability of the secondary processor to test the applications processor interface card was demonstrated.
5.2.7 Secondary processor system data bus controller card test

(a) Demonstration Procedure

(i) The console and the system were powered up.
(ii) "D" and "K" were selected in order to invoke the secondary processor test.
(iii) "I" was selected in order to invoke the system data bus controller card test.

(b) Result

(i) The secondary processor invoked the system data bus self-test, tested the dual port RAM on the card and returned the result to the RS232 terminal.

(c) Result Interpretation

The ability of the secondary processor to test the system data bus controller card was demonstrated.

5.2.8 Secondary processor mass storage controller card test

(a) Demonstration Procedure

(i) The console and the system were powered up.
(ii) "D" and "K" were selected in order to invoke the secondary processor test.
(iii) "J" was selected in order to invoke the mass storage controller card test.

(b) Result

(i) The primary processor invoked a self-test and tested the dual port RAM of the mass storage controller card. The result of the test was displayed on the RS232 terminal.

(c) Result Interpretation

The ability of primary processor to test the mass storage controller card was demonstrated.
6 REFERENCES

[LAW-BROWN, 1990]

[MEHTA, 1990]

[POLMANS, 1990]