TECHNICAL REPORT CR-RD-SS-86-4

ADDASET EVALUATION/CALIBRATION SUPPORT

Prepared By:
Peter L. Romine
Timothy A. Palmer
Terry N. Long
School of Engineering
The University of Alabama in Huntsville
Huntsville, AL 35899

Prepared For:
Systems Simulation and Development Directorate
Research, Development, and Engineering Center
US Army MICOM

JANUARY 1987

U.S. ARMY MISSILE COMMAND
Redstone Arsenal, Alabama 35898-5000

Cleared for public release; distribution is unlimited.
DISPOSITION INSTRUCTIONS

DESTROY THIS REPORT WHEN IT IS NO LONGER NEEDED. DO NOT RETURN IT TO THE ORIGINATOR.

DISCLAIMER

THE FINDINGS IN THIS REPORT ARE NOT TO BE CONSTRUED AS AN OFFICIAL DEPARTMENT OF THE ARMY POSITION UNLESS SO DESIGNATED BY OTHER AUTHORIZED DOCUMENTS.

TRADE NAMES

USE OF TRADE NAMES OR MANUFACTURERS IN THIS REPORT DOES NOT CONSTITUTE AN OFFICIAL ENDORSEMENT OR APPROVAL OF THE USE OF SUCH COMMERCIAL HARDWARE OR SOFTWARE.
ADDASET Evaluation/Calibration Support

Peter L. Romine, Timothy A. Palmer, and Terry N. Long

Verification and calibration software for the Air Defense Digital Avionics Seeker Enhancement Technology (ADDASET) hardware is documented. Software rationale, operation sequence, program listings and user aids are provided.

ADDASET DEBUG

Digital Avionics Program Listings Seeker User Aids
ADDASET EVALUATION/CALIBRATION SUPPORT

Final Technical Report for Period
28 November 1984 through 31 May 1985

January 1987

Prepared By:
Peter L. Romine
Timothy A. Palmer
Terry N. Long
School of Engineering
The University of Alabama in Huntsville
Huntsville, AL 35899

Prepared For:
Systems Simulation and Development Directorate
Research, Development, and Engineering Center
US Army MICOM

Cleared for public release; distribution is unlimited.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I.</td>
<td>INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>II.</td>
<td>DEBUG TEST PROGRAM</td>
<td>1</td>
</tr>
<tr>
<td>A.</td>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>B.</td>
<td>Discrete I/O Control</td>
<td>2</td>
</tr>
<tr>
<td>C.</td>
<td>Analog I/O Control</td>
<td>2</td>
</tr>
<tr>
<td>D.</td>
<td>Calibrate Analog-Buffer Card</td>
<td>2</td>
</tr>
<tr>
<td>E.</td>
<td>DEBUG Software Notes</td>
<td>3</td>
</tr>
<tr>
<td>III.</td>
<td>ADDITIONAL I/O TEST PROGRAMS</td>
<td>3</td>
</tr>
<tr>
<td>IV.</td>
<td>EVALUATION/CALIBRATION AIDS</td>
<td>4</td>
</tr>
<tr>
<td>V.</td>
<td>CONCLUSIONS AND RECOMMENDATIONS</td>
<td>7</td>
</tr>
</tbody>
</table>

APPENDIX A. DEBUG. A-1
APPENDIX B. ADDISP. B-1
APPENDIX C. DARAMP. C-1
APPENDIX D. ADTODA. D-1
APPENDIX E. DSCOUT. E-1
APPENDIX F. ACTIVITY. F-1

v/(vi Blank)
This technical report was prepared by the Research Staff of the Electrical and Computer Engineering Department, School of Engineering, The University of Alabama in Huntsville. The work documented in this report was performed by Terry N. Long, Timothy A. Palmer, and Peter L. Romine. The purpose of this report is to provide documentation of technical work performed and results obtained under delivery order number 0044, contract number DAAH01-82-D-A008. Mr. Terry N. Long was the principal investigator; Dr. M. M. Hallum, III, Chief, Systems Evaluation Branch, was the technical monitor; and Mr. Mark Horton also from the Systems Evaluation Branch of the Systems Simulation and Development Directorate, U.S. Army Missile Command, provided technical coordination.

The technical viewpoints, opinions, and conclusions expressed in this document are those of the authors and do not necessarily express or imply policies or positions of the U.S. Army Missile Command.
I. INTRODUCTION

A comprehensive test routine was required for verification and calibration of the Air Defense Digital Avionics Seeker Enhancement Technology (ADDASET) hardware built by the Boeing Company. A C-routine, DEBUG, and its supporting routines provide calibration and debug capabilities for the HAWK/ADDASET test bed. The programs were generated using a Zilog System-8000 UNIX-based development system which was chosen for compatibility with the Z8002-based single board computers used in the ADDASET system. Compatibility is such that compiled object modules and executables originating on the System-8000 can be transported directly to the computers in the ADDASET system.

Operation of the DEBUG test program, which is a very versatile multipurpose program, is described in Section II. Additional special purpose routines were developed during the hardware acceptance phase of the ADDASET program and are briefly described in Section III. Several user aids were also developed or identified during the acceptance process, and are described in Section IV. Conclusions and recommendations are presented in Section V.

II. DEBUG TEST PROGRAM

A. Introduction

The DEBUG test program is a versatile, multipurpose diagnostic program that can be used for operational verification or calibration of ADDASET hardware. Program execution is initiated after entering "debug". The program is fully menu-driven with the menu illustrated in Table 1 being displayed initially. As shown by the menu, five separate processes can be performed. The desired process can be chosen by the proper selection. Output and input discretes can be controlled as well as analog-to-digital converters (A/D's) and digital-to-analog converters (D/A's). There is also a routine to facilitate calibration of the system's analog buffers.

Discrete I/O control, analog I/O control, and analog buffer card calibration are described below. Additionally, some notes concerning DEBUG software design are provided.

<table>
<thead>
<tr>
<th>TABLE 1. Opening Menu</th>
</tr>
</thead>
<tbody>
<tr>
<td>O) Output byte to discrete channel</td>
</tr>
<tr>
<td>I) Input byte from discrete channel</td>
</tr>
<tr>
<td>A) A/D input routine</td>
</tr>
<tr>
<td>D) D/A output routine</td>
</tr>
<tr>
<td>C) Calibrate analog-buffer card</td>
</tr>
<tr>
<td>E) Exit to monitor</td>
</tr>
</tbody>
</table>

Enter selection (O,I,A,D,C,E):
B. Discrete I/O Control

The ADDASET system currently supports three 8-bit discrete input and three 8-bit discrete output channels. The digital interface is required for monitoring and controlling the four binary signals originating on the missile (Radar Enable, End of Sustain, Pitch Precession, and Antenna Center commands) and the eight software emulated signals (Radar Enable, Head Enable, Elevon Enable, Test Mode, Test Select 1, Test Select 2, Analog Stabilization Loop Closure, and Antenna Center Mode). Each discrete has a corresponding LED indicator on the front panel of the interface rack to display the status of each bit. By selecting the Input/Output byte option from the main menu, the operator can either read discrete input channel A or specify an arbitrary bit pattern for output on a selected channel.

Press "0" or "0" to output a byte on a discrete channel. The system will prompt the user for the channel number (0-2) and the data in hex (0-FF). After entry, the output data is also echoed to the terminal screen.

Press "I" or "i" to input data from discrete channel A. Input data is displayed on the terminal screen and updated continuously. The input operation can be halted at any time by pressing the ESC key on the terminal keyboard.

C. Analog I/O Control

The A/D and D/A options allow the operator to monitor the voltage at any of 24 analog inputs or specify an arbitrary voltage for output on any of 16 analog outputs. Voltages are displayed on the terminal screen in floating point. Real valued voltages can also be entered at the keyboard for output on a selected analog line.

Press "A" or "a" to convert an external analog voltage signal to digital for display on the terminal screen. The operator is prompted to select an analog input channel (0-23). The voltage is then displayed on the terminal screen and updated continuously until the ESC key is pressed.

Press "D" or "d" to output an arbitrary voltage. The operator is prompted to enter the channel number (0-15) and the output data. Output data can be any real number voltage within the limits of the power supply (+10V). The actual output value is echoed to the terminal after output on the selected channel.

D. Calibrate Analog-Buffer Card

The analog-buffer card is the analog interface between the missile and the ADDASET rack. The buffer card's primary function is for adjustment of biases and gains associated with each analog I/O line on the interface rack. It also provides protection for missile and rack components. To aid in calibration of the analog-buffer card, the calibration option provides the menu shown in Table 2.
TABLE 2. Analog-Buffer Card Calibration Menu

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>G)</td>
<td>select ground</td>
</tr>
<tr>
<td>+)</td>
<td>select plus(+) voltage</td>
</tr>
<tr>
<td>-)</td>
<td>select minus(−) voltage</td>
</tr>
<tr>
<td>D)</td>
<td>display A/D channels</td>
</tr>
<tr>
<td>M)</td>
<td>main menu</td>
</tr>
</tbody>
</table>

Select ground, plus voltage, and minus voltage options apply the corresponding signal to each of the analog inputs. The converted voltage can then be monitored using the display A/D function. In this configuration, the analog card can be calibrated by adjusting the gains and/or biases on the card until the correct voltage is displayed on the terminal. By pressing "G", "+", or "−", the inputs to the buffer card are either grounded, set to the precision positive calibration voltage, or set to the precision negative calibration voltage.

To monitor a particular channel, press "D" or "d" and input the desired A/D channel number (0-15) when prompted. The voltage presented to the selected channel is displayed on the terminal screen and is updated continuously until the ESC key is pressed. The escape key returns the user to the configuration menu.

Press "M" or "m" to exit the calibration menu and return to the opening menu.

E. DEBUG Software Notes

A listing of DEBUG code is presented in Appendix A. Some important pieces of information can be derived from this code. First, the program's menu format can be examined by examining the arguments of the "mess-c" functions. Also, the register addresses for I/O and control functions can be easily found in the code.

A large portion of the code was developed to facilitate manipulation of floating point numbers. Some of the routines are directly callable from C and some are not. It is useful to note that they can be used for a variety of applications.

III. ADDITIONAL I/O TEST PROGRAMS

DEBUG resulted from the combination of a number of test and calibration programs. Several other routines were developed in addition to those included in DEBUG and they perform a variety of useful functions.
1. ADDISP

ADDISP is particularly useful because it provides for the simultaneous display of up to seven of the 23 A/D channels. The user must supply the number of channels to be displayed at one time (1-7, do not choose 0). The user must also supply which ADC (1-23) should be displayed in each column (0-6) of the display. The user can then choose continuous updates (press C), discrete updates (press D), or pause (press P). A listing of ADDISP is provided in Appendix B.

2. DARAMP

DARAMP is also very useful as a verification tool. It generates a ramp output to a block of D/A's. The user must supply the start channel and stop channel of the block of D/A's to be exercised as well as the start value and stop value of the ramp. The user must also supply the step size of the ramp input divided by sixteen. Dynamic operation of the D/A's can be examined by using this routine, and each discrete level can be evaluated. A listing of DARAMP is provided in Appendix C.

3. ADTODA

ADTODA facilitates simultaneous output to a D/A channel from an A/D channel. This permits operation verification by tying pairs of channels together. A listing of ADTODA is provided in Appendix D.

4. DSOUT

DSOUT simply sequences all of the output discretes. The program is marginally useful, and its listing is provided in Appendix E.

5. ACTIVITY

ACTIVITY exercises the operation modes. It is marginally useful, and a listing of the program is provided in Appendix F.

IV. EVALUATION/CALIBRATION AIDS

A detailed review of ADDASET documentation revealed that wiring information exists for all system interfaces. However, the information exists in various forms with no way to easily trace a signal from beginning to end. Figure 1 was developed in an effort to summarize the configuration and to provide a starting point for the wiring documentation. ADDASET/HAWK system elements are identified along with individual connectors. Each connector has been assigned a number, and they are provided in the figure. Additionally, the types of wiring and the numbers of lines have been identified. The most useful characteristic of the figure is the identification of the documentation that corresponds to each interface. The amount of interface documented by each piece of documentation and the set numbers of the corresponding documentation are also noted.
Figure 1. ADDASET wiring illustration.
Table 3. A/D and D/A Scaling

<table>
<thead>
<tr>
<th>Hex</th>
<th>Volts</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>7FF0</td>
<td>+10</td>
<td>2047</td>
</tr>
<tr>
<td>7320</td>
<td>+9</td>
<td>1842</td>
</tr>
<tr>
<td>6660</td>
<td>+8</td>
<td>1638</td>
</tr>
<tr>
<td>5990</td>
<td>+7</td>
<td>1433</td>
</tr>
<tr>
<td>4CC0</td>
<td>+6</td>
<td>1228</td>
</tr>
<tr>
<td>3FF0</td>
<td>+5</td>
<td>1023</td>
</tr>
<tr>
<td>3330</td>
<td>+4</td>
<td>0819</td>
</tr>
<tr>
<td>2660</td>
<td>+3</td>
<td>0614</td>
</tr>
<tr>
<td>1990</td>
<td>+2</td>
<td>0409</td>
</tr>
<tr>
<td>0CD0</td>
<td>+1</td>
<td>0205</td>
</tr>
<tr>
<td>0000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>F330</td>
<td>-1</td>
<td>-0205</td>
</tr>
<tr>
<td>E670</td>
<td>-2</td>
<td>-0409</td>
</tr>
<tr>
<td>D9A0</td>
<td>-3</td>
<td>-0614</td>
</tr>
<tr>
<td>CCD0</td>
<td>-4</td>
<td>-0819</td>
</tr>
<tr>
<td>C010</td>
<td>-5</td>
<td>-1023</td>
</tr>
<tr>
<td>B340</td>
<td>-6</td>
<td>-1228</td>
</tr>
<tr>
<td>A670</td>
<td>-7</td>
<td>-1433</td>
</tr>
<tr>
<td>99A0</td>
<td>-8</td>
<td>-1638</td>
</tr>
<tr>
<td>8CE0</td>
<td>-9</td>
<td>-1842</td>
</tr>
<tr>
<td>8010</td>
<td>-10</td>
<td>-2047</td>
</tr>
</tbody>
</table>
Another verification aid is provided in Table 3. It contains scaling numbers used by the A/D's and D/A's. However, its usefulness may be marginal since scaling is now provided within DEBUG code.

V. CONCLUSIONS AND RECOMMENDATIONS

The programs and aids described in this report provide a comprehensive means to verify ADDASET hardware operation and provide ADDASET calibration. They have been used in the acceptance testing phase of the ADDASET program and have been used to calibrate the analog components of the system. They have also been used to monitor missile discretes in the various modes of operation.

The C language used for program development has proved to be very advantageous for I/O type activities. Therefore, it is recommended that the language be used for all ADDASET programming which requires user or hardware interfaces.

The System-8000 development system used for program development has presented several difficulties. Consequently, it is recommended for alternative systems to be examined for program development. PC type systems should be carefully considered.

The ADDASET program now has the test bed required to begin the development of digital avionics techniques and the aids required to maintain that test bed. Therefore, the ADDASET program should proceed at a rapid pace.
APPENDIX A

DEBUG

A-1/(A-2 Blank)
Main program to allow user to modify D/A and output discrete as monitor A/D and input discrete.

Functions defined in debug.c module:

```c
C routine provides for the user interface.

Externals

```
mess c("O - output byte to discrete channel\n\r");
mess c("I - input byte from discrete channel\n\r");
mess c("A - A/D input routine\n\r");
mess c("D - D/A output routine\n\r");
mess c("C - Calibrate analog-buffer card\n\r");
mess c("E - Exit to monitor\n\r");

ok = 'N';
while ( ok != 'Y') {
   out_crlf();
   mess c("Enter selection (O,I,A,D,C,E) :");
   c = wait_char();
   if ((c == 'O') | (c == 'I') | (c == 'A') | (c == 'D')) ok = 'Y'
   if ((c == 'C') | (c == 'E')) ok = 'Y';
}

out_crlf();
switch(c) {
   case 'O':
      out_crlf();
      mess c("Enter discrete channel number (0-2) :"
      ok = 'N';
      while ( ok != 'Y') {
         chan = in_dec();
         if ((chan <= 2) & (chan >= 0)) ok = 'Y'
         if (ok == 'N')
            out_char(0x07);
            mess c("Invalid channel. ");
            mess c("Try again ");
            out_crlf();
            } /* if */
   out_crlf();
   ok = 'N';
   while (ok != 'Y') {
      mess c("Enter channel ");
      out_int(chan);
      mess c(" data (0-FF) :");
      data = in_int();
      if ((data <= 0) & (data <= 256)) ok = 'Y'
   } /* while */
   out_crlf();
   switch (chan) {
      case 0: cio_outa(data);
      break;
      case 1: cio_outb(data);
      break;
      case 2: cio_outc(data);
      break;
      default: break;
   } /* switch */
out_crlf();
mess_c("Data ("");
out_int(data);
mess_c(") present on channel ");
out_int(chan);
out_crlf();
break;
}

case 'I': {
    out_crlf();
    out_crlf();
mess_c("Press ESCape to exit ");
    out_crlf();
    out_crlf();
mess_c("Input discrete data ");
    out_crlf();
mess_c("b0 b1 b2 b3 b4 b5 b6 b7");
    out_crlf();
    while ((c=in_char()) != 0x1b) {
        data = cio_ina();
        out_char(0x0d);
        j=1;
        while ( (j <= 128) ) {
            k = (j & data);
            if ( (k == 0) ) mess_c("0 ");
            if ( (k != 0) ) mess_c("1 ");
            j *= 2;
        }
    }
    out_crlf();
    break;
}

case 'A': {
    out_crlf();
    ok='N';
    while ( ok != 'Y' ) {
        mess_c("Enter channel number (0-23) ":
        chan=in_dec();
        if ( ((chan <= 23) & (chan >= 0)) ) ok='Y'
    } /* while */
    out_crlf();
mess_c("Press ESCape to exit ");
    out_crlf();
    out_crlf();
mess_c("A/0 channel ");
    out_int(chan);
    out_crlf();
}
while ((c-in_char()) != 0x1b) {
    point=array;
    ad_in_a(point,1);
    point=array+12;
    ad_in_b(point,1);
    data=array[chan];
    out_char(0x0d);
    out_int(data);
    long2=data;
    out_char(' ');
    fp_out(fdiv(fpconv(long2),con32768p8));
}
out_crlf();
break;
}
case 'D' :
    out_crlf();
    ok = 'N';
    while (ok != 'Y') {
        mess_c("Enter channel number (0-15) : ");
        chan=in_dec();
        out_crlf();
        if ((chan >= 0) & (chan <= 15)) ok = 'Y'
        if (ok == 'N') {
            out_char(0x07);
            mess_c("Invalid channel number ");
            mess_c("Try again");
            out_crlf();
        }
        out_crlf();
        out_crlf();
        mess_c("Enter data for channel ");
        out_int(chan);
        mess_c(" ");
        data=in_int();
        array[0]=data;
        point=array;
        da_out(point,chan,1);
        mess_c("Data ");
        out_int(array[0]);
        mess_c(" present on channel ");
        out_int(chan);
        out_crlf();
        break;
    }
case 'C' : {
    while(1) {
        mess_c("\n\n\r");
        mess_c("Analog buffer card calibration \n\r");
        mess_c("G - select ground\n\r");
        mess_c("+ - select plus voltage\n\r");
        mess_c("- - select minus voltage\n\r");
        mess_c("D - display A/D channels\n\r");
        mess_c("M - main menu");
        c = wait_char();
        if (c == 'M') break;
        switch (c) {
        case 'G': {
            cio_outa(0x48);
            mess_c("\n\rTest mode: Ground\n\rbreak;
        }.
        case '+' : {
            cio_outa(0x78);
            mess_c("\n\rTest mode: Plus\n\rbreak;
        }
        case '-' : {
            cio_outa(0x58);
            mess_c("\n\rTest mode: Minus\n\rbreak;
        }
        case 'D' : {
            mess_c("\n\rChannel :");
            i=in_dec();
            if (i > 15) {
                mess_c("\n\rToo big\n\rbreak;
            }
            if (i < 0) {
                mess_c("\n\rToo small\n\rbreak;
            }
        mess_c("\n\rPress ESCape to exit.\nWhile (((c=in_char()) != 0x1b)
point=array;
ad_in_a(point,1);
point=point+12;
ad_in_b(point,1);
data=point[i];
long2=data;
long2=fpconv(long2);
long2=fdiv(long2,con32
fp_out(long2);
out_char(0x0d);
} /* while */
break;
} /* case 'D' */
*/ switch */
  } /* while */
break;
} /* case 'C' */
default: go='N';
} /* switch */
} /* while */
} /* main */
Functions defined in inout_c.s module:

**in_char**
Assembly routine to check whether a character available in the receive buffer register of the UART from the keyboard.
ENTRY: none
EXIT: RLO = 0, if no character was available
       RLO = character, otherwise

**wait_char**
Assembly routine to wait for a character to be available from the keyboard.
ENTRY: none
EXIT: RLO = character

**out_char**
Assembly routine to transmit one character to console port (terminal).
ENTRY: RL7 = character to output
EXIT: none

**out_crlf**
Assembly routine to transmit a <CR><LF> to the console port (terminal)
ENTRY: none
EXIT: none

**mess_c**
Assembly routine to transmit a sequence of null-terminated characters to the console port (terminal)
ENTRY: R7 = address of start of string buffer
EXIT: none

**out_int**
Assembly routine to convert a 16-bit integer to ASCII-hex and send the characters to the console port (terminal).
ENTRY: R7 = 16-bit integer to print
EXIT: none

**in_int**
Assembly routine to accept a 16-bit hex integer constant from the keyboard. The characters are echoed to the console port (terminal) as they input. Any out-of-band characters cause an error message to be displayed and the conversion is again.
ENTRY: none
EXIT: R2 = 16-bit result

Externals (NONE)
_inout_c MODULE
!
RAM ALLOCATION CONSTANTS!

CONSTANT
!
SCC REGISTER ADDRESSES!
SCCOA := %FC21
SCCDA := %FC31

GLOBAL

_in_char PROCEDURE
ENTRY

! INPUT A CHARACTER FROM THE TERMINAL!

TINPUT: INB RLO,SCCOA
BITB RLO,$0
JR Z,TDONE
INB RLO,SCCDA
RESB RLO,$7
LDB RL2,RLO
CLRB RH2
RET
TDONE: CLR R2
RET
END_in_char

_wait_char PROCEDURE
ENTRY

! INPUT A CHARACTER FROM THE TERMINAL!

TWAIT: INB RLO,SCCOA
BITB RLO,$0
JR Z,TWAIT
INB RLO,SCCDA
RESB RLO,$7
LDB RL2,RLO
CLRB RH2
RET
END_wait_char
_out_char PROCEDURE
ENTRY
LDB RL0,RL7

! OUTPUT A CHARACTER TO THE TERMINAL!

TOUTCH: PUSH @R15,R1
LDAR R1,$+6
JR TOCHNS
POP R1,®R15
RET

! OUTPUT CHARACTER TO TERMINAL WITHOUT USING RAM!

TOCHNS: INB RH0,SCCOA
BITB RH0,#2
JR Z,TOCHNS
TOUT10: OUTB SCCDA,RL0
JP ®R1
END _out_char

$out_crlf$ PROCEDURE
ENTRY
LDB RL0,'%R'
CALR TOUTCH
LDB RL0,'%L'
JR TOUTCH
END $out_crlf$

_mess_c $PROCEDURE$
ENTRY
print2: ld r6,®r7
inc r7,#2
ldb r11,rh6
cpb r11,#0
ret z
calr print3
ldb r11,r16
cpb r11,#0
ret z
calr print3
jp print2
print3: inb rh0,SCCOA
bitb rh0,#2
jr Z,print3
outb SCCDA,r11
ret
END $mess_c$
!out_int PROCEDURE
ENTRY
    ldb    r11,rh7    
call   out_byte    
    ldb    r11,r17   
call   out_byte    
    ret     out_byte:
        ldb    r10,r11  
        andb   r10,#%0f0 
        stlb    r10,4   
        call   out_nib  
        ldb    r10,r11  
        andb   r10,#%0f  
        call   out_nib  
        ret     out_nib:
            addb   r10,#%030  
            cpb    r10,#%3a  
            jr     c,out_nib2  
            addb   r10,#%07  
    out_nib2:
        inb    rh0,SCCOA  
        bitb   rh0,#2    
        jr     z,out_nib2  
        outb   SCCDA,r10  
        ret     
END_out_int
in_int PROCEDURE
ENTRY

! this routine accepts a 16 bit integer (unsigned) from the console!
! and converts it to binary!

ii0: clr r8
    clr r9

    clr rS ! clear the result registers!
    clr r9

ii1: call _wait_char
    ld r7,r2
    call _out_char
    cpb F12,$%0d
    jr nz,ii2
    ld r2,r8
    ret

    ii2: call atob
        jr c,ii3
        add rS,r2
        jr nz,ii2
        ret

    iil: call _wait_char
        ld r7,r2
        call _out_char
        cpb F12,$%0d
        jr nz,ii2
        ld r2,r8
        ret

    ii3: lda r7,II4
        call _mess_c
        jp ii0

    ii4: ARRAY[*BYTE] := '?REDO%R%L%00'

END _inout_c

A-13
Functions defined in rtc.s module:

cio_init
Assembly routine to initialize the CIO chips which are responsible for Real-Time-Clock (RTC) and discrete (bit) I/O to the Digital Avionics interface rack. The three discrete ports on the CIO chip are initialized as follows:
- PORTA input only
- PORTB input only
- PORTC input only

The three ports on the second CIO chip are initialized as follows:
- PORTA output only
- PORTB output only
- PORTC output only

(see the CIO chip manual for more information)

ENTRY: none
EXIT: none

rtc_init
Assembly language routine which is installed a interrupt vector for RTC interrupts. The routine calls the head_tracking_stabilization loop and returns. This routine is installed in the PSAP the Z8000 as an optional function of CIO_INIT

ENTRY: none (interrupt)
EXIT: none

cio_ina, cio_outa, cio_inb, cio_outb, cio_outc
Assembly language routines to input-from/output-to the CIO chips. All cio_inx routines acquire input from the first CIO chip, port x. All cio_outx routines send output to the second CIO chip

ENTRY: RL7 = data to output (for output, or none, for input)
EXIT: RL2 = none (for output, or 8-bit data, for input)

Externals  ( mess_c, out_char, out_int, out_crlf, head, Cycle_Cntr )
**_cio_control MODULE_**

**CONSTANT**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CI01</td>
<td>%fd01</td>
<td>base address of CIO-1</td>
</tr>
<tr>
<td>MICR1</td>
<td>%fd02</td>
<td>master interrupt control register</td>
</tr>
<tr>
<td>MCCR1</td>
<td>CI01+2</td>
<td>master configuration control register</td>
</tr>
<tr>
<td>CTCSR11</td>
<td>CI01+20</td>
<td>Counter/Timer-1 command and status register</td>
</tr>
<tr>
<td>CTCSR12</td>
<td>CI01+22</td>
<td>&quot;</td>
</tr>
<tr>
<td>CTMSR11</td>
<td>CI01+56</td>
<td>Counter/Timer-1 mode specification register</td>
</tr>
<tr>
<td>CTMSR12</td>
<td>CI01+58</td>
<td>Counter/Timer-2 mode specification register</td>
</tr>
<tr>
<td>CTCCR1L1</td>
<td>CI01+34</td>
<td>Counter/Timer-1 current count MSB</td>
</tr>
<tr>
<td>CTCCR1L2</td>
<td>CI01+36</td>
<td>Counter/Timer-2 current count MSB</td>
</tr>
<tr>
<td>CTCCR1H1</td>
<td>CI01+32</td>
<td>Counter/Timer-1 current count LSB</td>
</tr>
<tr>
<td>CTCCR1H2</td>
<td>CI01+38</td>
<td>Counter/Timer-2 current count LSB</td>
</tr>
<tr>
<td>CTTCR1L1</td>
<td>CI0+46</td>
<td>Counter/Timer-1 time constant register LSB</td>
</tr>
<tr>
<td>CTTCR1H1</td>
<td>CI0+44</td>
<td>Counter/Timer-1 time constant register MSB</td>
</tr>
<tr>
<td>CTTCR1L2</td>
<td>CI0+50</td>
<td>Counter/Timer-2 time constant register LSB</td>
</tr>
<tr>
<td>CTTCR1H2</td>
<td>CI0+48</td>
<td>Counter/Timer-2 time constant register MSB</td>
</tr>
<tr>
<td>CTIVR</td>
<td>CI0+8</td>
<td>Counter/Timer-1 interrupt vector register</td>
</tr>
</tbody>
</table>

! CIO REGISTER ADDRESSES !

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IVR1A</td>
<td>%fd05</td>
<td>cio 1, port A, interrupt vector register</td>
</tr>
<tr>
<td>IVR1B</td>
<td>%fd07</td>
<td>cio 1, port B, interrupt vector register</td>
</tr>
<tr>
<td>IVR1CT</td>
<td>%fd09</td>
<td>cio 1, counter/timer interrupt vector register</td>
</tr>
<tr>
<td>DPPR1C</td>
<td>%fd0B</td>
<td>cio 1, port C, data path polarity register</td>
</tr>
<tr>
<td>DDR1C</td>
<td>%fd0D</td>
<td>cio 1, port C, data direction register</td>
</tr>
<tr>
<td>SIOCR1C</td>
<td>%fd0F</td>
<td>cio 1, port C, special I/O register</td>
</tr>
<tr>
<td>PCSR1A</td>
<td>%fd11</td>
<td>cio 1, port A, command and status register</td>
</tr>
<tr>
<td>PCSR1B</td>
<td>%fd13</td>
<td>cio 1, port B, command and status register</td>
</tr>
<tr>
<td>PDR1A</td>
<td>%fd1B</td>
<td>cio 1, port A, data register</td>
</tr>
<tr>
<td>PDR1B</td>
<td>%fd1D</td>
<td>cio 1, port B, data register</td>
</tr>
<tr>
<td>PCDR1</td>
<td>%fd1F</td>
<td>cio 1, port C, data register</td>
</tr>
<tr>
<td>CVR1</td>
<td>%fd3F</td>
<td>cio 1, counter/timer current vector</td>
</tr>
<tr>
<td>PMSR1A</td>
<td>%fd41</td>
<td>cio 1, port A, mode specification register</td>
</tr>
<tr>
<td>PHSR1A</td>
<td>%fd43</td>
<td>cio 1, port A, handshake specification register</td>
</tr>
<tr>
<td>DPPR1A</td>
<td>%fd45</td>
<td>cio 1, port A, data patch polarity register</td>
</tr>
<tr>
<td>DDR1A</td>
<td>%fd47</td>
<td>cio 1, port A, data direction register</td>
</tr>
<tr>
<td>SIOCR1A</td>
<td>%fd49</td>
<td>cio 1, port A, special I/O register</td>
</tr>
<tr>
<td>PMSR1B</td>
<td>%fd51</td>
<td>cio 1, port B, mode specification register</td>
</tr>
<tr>
<td>PHSR1B</td>
<td>%fd53</td>
<td>cio 1, port B, handshake specification register</td>
</tr>
<tr>
<td>DPPR1B</td>
<td>%fd55</td>
<td>cio 1, port B, data patch polarity register</td>
</tr>
<tr>
<td>DDR1B</td>
<td>%fd57</td>
<td>cio 1, port B, data direction register</td>
</tr>
<tr>
<td>SIOCR1B</td>
<td>%fd59</td>
<td>cio 1, port B, special I/O register</td>
</tr>
<tr>
<td>MICR2</td>
<td>%fe01</td>
<td>same as CIO-1 (above)</td>
</tr>
<tr>
<td>MCCR2</td>
<td>%fe03</td>
<td></td>
</tr>
<tr>
<td>IVR2A</td>
<td>%fe05</td>
<td></td>
</tr>
<tr>
<td>IVR2B</td>
<td>%fe07</td>
<td></td>
</tr>
<tr>
<td>IVR2CT</td>
<td>%fe09</td>
<td></td>
</tr>
<tr>
<td>DPPR2C</td>
<td>%fe0B</td>
<td></td>
</tr>
<tr>
<td>DDR2C</td>
<td>%fe0D</td>
<td></td>
</tr>
<tr>
<td>SIOCR2C</td>
<td>%fe0F</td>
<td></td>
</tr>
</tbody>
</table>
ENTRY

! on entry r7 contains the desired time constant!

! first initialize the interrupt stuff!

di vi,nvi    ! disable vi,nvi!

ldb r10,%01  ! do a Cio reset!
outb MICR1,r10 ! by setting the reset bit!

ldctl r1,psapoff ! current PSAP (should be %FD00)!

add r1,%1C    ! offset to FCW location!
ld r0,%4000   ! setup FCW for vectored interrupts!
ld @r1,r0    ! store %4000 at xx1c in PSAP!

add r1,%02    ! add to get next address!
ld @r1,_rtc_int ! set vector to our guy!

push @r15,r7    ! save r7 just in case!
push @r15,r6    ! save r6 just in case!
! set MIE (mast. int. en)!
outb MICR1, r10

! put back out!

! enable sqw wave, continuous cycle!
outb CTRMS11, r10
outb CTRMS12, r10

! put it back now!

! zero the interrupt vector!
outb CTRVR, r10

! get time constant for CT-2!
pop r6, @r15
outb CTTCR1L2, r16
outb CTTCR1H2, r16

! set up CTTCR1L2!

! set up time constant for CT-1!
outb CTTCR1L1, r17
outb CTTCR1H1, r17

! lsh!

! msh!

! set IE for Counte/Timer 2!
outb CTCR12, r10

! and same for C/T 2!

! clear IE for C/T 1!
outb CTCR11, r10

! enable (timer-1 clocks timer-2),!
! ports A, B, and C of unit 1!
outb MCCR1, r10

! using MCCR!

! unit 1 initialization!
! set MSR Port A & B to input single buffer!

outb PMSR1, r10
outb PMSR1A, r10
outb PMSR1B, r10

! clear PCSR's, DPPR's, and SIOCR's!
clr r0
outb PCSR1, r10
outb PCSR1A, r10
outb PCSR1B, r10
outb DPPRA1, r10
outb DPPRB1, r10
outb DPPRC1, r10
outb SIOCR1A, r10
outb SIOCR1B, r10
outb SIOCR1C, r10

! set Data Direction for input!

outb DDR1, r10
outb DDR1A, r10
outb DDR1B, r10
outb DDR1C, r10

A-17
! unit 2 initialization!
! clear the reset bit!

clr r0
outb MICR2,r10

! set MSR Port A & B to output single buffered!

ldb r10,#%10
outb FMSR2A,r10
outb FMSR2B,r10

! clear PCSR's, DPFR's, and SIOCR's!

clr r0
outb PCSR2A,r10
outb PCSR2B,r10
outb DPFR2A,r10
outb DPFR2B,r10
outb DPFR2C,r10
outb SIOCR2A,r10
outb SIOCR2B,r10
outb SIOCR2C,r10

! set Data Direction for output!

outb DDR2A,r10
outb DDR2B,r10
outb DDR2C,r10

! enable ports A, B, & C!

ldb r10,#%94
outb MCCR2,r10

ldb r10,#%80 ! enable master interrupts!
outb MICR1,r10

! set trigger and gate bits C/T 1!
outb CTCSR11,r10
outb CTCSR12,r10

! and C/T 2!

ei vi,nvi
ret ! end of initialization!

END _cio_init
_rtc_int PROCEDURE
ENTRY
! routine to service the rtc interrupt(s) !

ldm   regs_save,r0,#15   ! save the context at entry !
ldb   r10,#%26
outb  CTCSR12,r10
!   call   _head
!   inc   _Cycle_Cntr
ldm   r0,regs_save,#15   ! restore context !
iret   ! return from interrupt !

END _rtc_int
regs_save: ARRAY [16 WORD]

_cio_ina PROCEDURE
ENTRY
! input port A !
inb    r12,PDR1A
ret
END _cio_ina

_cio_outa PROCEDURE
ENTRY
! output port A !
outb   PDR2A,r17
ret
END _cio_outa

_cio_inb PROCEDURE
ENTRY
! input port B !
inb    r12,PDR1B
ret
END _cio_inb
cio_outb PROCEDURE
ENTRY
! output port B !
  outb PDR2B,r17
  ret
END _cio_outb

cio_outc PROCEDURE
ENTRY
! output port C !
  outb PCDR2,r17
  ret
END _cio_outc

END _cio_control
Functions defined in ad_da.s

ad_in_a  
Assembly language subroutine to acquire 12 bit data from first set of 12 A/D channels. All 12 channels are converted.
ENTRY:  R7 = address of data buffer (must contain at least 24 bytes of free space)
         R6 = 1 if first call, 0 if not
EXIT:  none

ad_in_b  
(Same as ad_in_a, except for channels 12-23)

da_out  
Assembly language subroutine to output 12-bit data to the D/A converter.
ENTRY:  R7 = address of data buffer
         R6 = Start channel (0-15)
         R5 = Number of channels (0-15)
EXIT:  none

Externals  ( cio_outc,cio_outb,cio_outc )

_ad_da MODULE

CONSTANT
! A_D INPUT BASE ADDRESS !

BASE1  :=  %f600
BASE2  :=  %f610
CSRL1  :=  BASE1
SCRL1  :=  BASE1+2
MARL1  :=  BASE1+4
DREG1  :=  BASE1+6
CSRL2  :=  BASE2
SCRL2  :=  BASE2+2
MARL2  :=  BASE2+4
DREG2  :=  BASE2+6
RDY    :=  15 ! because of byte swap !
PND    :=  2  ! when doing word I-O !
CAT    :=  1
FST    :=  0

! D_A OUTPUT CHANNEL BASE ADDRESS !

BASE    :=  %f710
CHAN0   :=  BASE
CHAN1   :=  BASE+2
CHAN2   :=  BASE+4
CHAN3   :=  BASE+6
CHAN4   :=  BASE+8
CHAN5   :=  BASE+10
CHAN6 := BASE + 12
CHAN7 := BASE + 14
BASE3 := f720
CHAN8 := BASE3
CHAN9 := BASE3 + 2
CHAN10 := BASE3 + 4
CHAN11 := BASE3 + 6
CHAN12 := BASE3 + 8
CHAN13 := BASE3 + 10
CHAN14 := BASE3 + 12
CHAN15 := BASE3 + 14

EXTERNAL
_cio_outc PROCEDURE
_cio_outb PROCEDURE
_cio_outa PROCEDURE

GLOBAL

_ad_in_a PROCEDURE
ENTRY
! r7 has the load address for data (passed from C) !
! r6 has: 1 if start/stop channels are to be updated !
! 0 if start/stop channels have already been initialized !
! check for first time thru :
ld r4, r6
! init bit !
cp r4, #1
! 1 means re-init !
jr z, ad07
! go if init desired !
in r1, CSRL1
bit r1, #PND
jr z, ad37
ad07: ld r0, %0100
! output init bit to control !
out CSRL1, r0
ad10: in r1, CSRL1
! wait for CAT to clear !
bit r1, #CAT
jr nz, ad10
ld r0, %0100
! write 1 to scan !
out CSRL1, r0
ad20: in r1, CSRL1
! wait for CAT to clear !
bit r1, #CAT
jr nz, ad20
ld r0, %0000
! write 0 to scan !
out CSRL1, r0
ad25: in r1, CSRL1
! wait for CAT to clear !
bit r1, #CAT
jr nz, ad25
ld r1, %0800
! set SCN bit in control !
out CSRL1, r1
ld r1, %0072
! set pacer to 100KH !
out MARL1, r1
ad30: in r1, CSRL1
! wait for scan to clear !
bit r1, #CAT
jr nz, ad30
ld r6, #0b
! do all 12 channels !
out SCRL1,r6
ad35:  in r1,CSRL1
bit r1,$CAT
jr nz,ad35
out CSRL1,r1
ad37:  ld r1,$0900
! wait for CAT to clear!
out CSRL1,r1
ad40:  in r1,CSRL1
bit r1,$C  r z,a! netiocolad4  ld r2,$80C
ld r3,r7
! set SCN and software trigger
ad41:  in r1,CSRL1
bit r1,$RDY
jr z,ad4f
in r0,DREG1
exb r0,r10
ld @r3,r0
inc r3,$2
djnz r2,ad401
! loop till all data read!
ret
! end of new section!
END _ad_in_a

_ad_in_b PROCEDURE
ENTRY
! check for first time thru!
ld r4,r6
! init bit!
cp r4,$1
! see if re-init desired!
jr z,ad4f
in r1,CSRL2
bit r1,$CAT
jr z,ad77
ad4f:  ld r0,$0010
! output INIT bit to control!
out CSRL2,r0
ad50:  in r1,CSRL2
bit r1,$CAT
jr nz,ad50
ld r0,$0100
! write 1 to scan!
out CSRL2,r0
ad60:  in r1,CSRL2
bit r1,$CAT
jr nz,ad60
ld r0,$0000
! write 0 to scan!
out CSRL2,r0
ad65:  in r1,CSRL2
bit r1,$CAT
jr nz,ad65
ld r0,$0800
! set SCN bit in control!
out CSRL2,r0
ld r1,$0072
! set pacer to 100KH!
out MARL2,r1
ad70:  in r1,CSRL2
bit r1,$CAT
! wait for CAT to clear!
jr nz,ad70
ld r6,##0b
out SCRL2,r6
ad75: in r1,CSRL2
bit r1,#CAT
jr nz,ad75
ad77: ld r1,##0900
out CSRL2,r1
ad80: in r1,CSRL2
bit r1,#CAT
jr nz,ad80

! new section!
ad800: ld r2,##0c
ld r3,r7
ad801: in r1,CSRL2
bit r1,#RDY
jr z,ad801
in r0,DREG2
exb rh0,r10
ld @r3,r0
inc r3,#2
djnz r2,ad801
ret
END _ad_in_b

! end of routine!
da_out PROCEDURE
ENTRY
! r7 has the data address, r6 has the start channel, r5 has the # chan
daloo: cp r5,#0
ret z
ld r1,r6
inc r6,#1
ld r2,@r7
inc r7,#2
call outda
inc r5,#1
call _cio outc
djnz raluop
ret

outda: cp r3,#3
jr gt,doit
push @r15,r7
ld r7,#2
call _cio outc
ld r7,r3
and r7,#3
call _cio outb
ld r7,#3
call _cio outc
doi: pop r7,@r15
sll r3,#1
ld r4,##710
sd r3,#1
add r4, r3      ! add offset to start address !
exb rh2, r12    ! exchange bytes for output !
out @r4, r2     ! output the word to the Dig-Analog co
ret             ! end of the routine !

END _da_out

END _ad_da
Functions defined in fpsubs.s module:

All of the below subroutines are used to manipulate floating point data.

+C"-interface routines

The following routines are directly callable from "C". A brief summary of the function and an example syntax is given below.

<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Description</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>fadd</td>
<td>add two floating point numbers</td>
<td>ex: fp_result = fadd(fp1,fp2);</td>
</tr>
<tr>
<td>fmul</td>
<td>multiply two floating point numbers</td>
<td>ex: fp_result = fmul(fp1,fp2);</td>
</tr>
<tr>
<td>fdiv</td>
<td>divide two floating point numbers (fp1/fp2)</td>
<td>ex: fp_result = fdiv(fp1,fp2);</td>
</tr>
<tr>
<td>fcmp</td>
<td>compare two floating point numbers and returns a longword result</td>
<td>ex: long_result = fcmp(fp1,fp2);</td>
</tr>
<tr>
<td>fsub</td>
<td>subtracts two floating point numbers</td>
<td>ex: fp_result = fsub(fp1,fp2);</td>
</tr>
<tr>
<td>fint</td>
<td>extracts the integer portion of a floating point number returns the result as a longword integer</td>
<td>ex: long_result = fint(fp1);</td>
</tr>
<tr>
<td>frac</td>
<td>extracts the fractional portion of a floating point number</td>
<td>ex: fp_result = frac(fp1);</td>
</tr>
<tr>
<td>fp_in</td>
<td>allows keyboard input of floating point number. The routine accepts standard forms for input (including exponential notation)</td>
<td>ex: fp_result = fp_in();</td>
</tr>
<tr>
<td>fp_out</td>
<td>displays a floating point number in non-exponential form</td>
<td>ex: fp_out(fp1);</td>
</tr>
<tr>
<td>fp_out_e</td>
<td>displays a floating point number in exponential form</td>
<td></td>
</tr>
<tr>
<td>fpconv</td>
<td>converts a long integer to floating point representation</td>
<td>ex: fp_result = fpconv(long_int);</td>
</tr>
</tbody>
</table>
fpcon obtains the floating point equivalent of a character s expression.
ex:     fp_result = fpcon("-10.256E-3");

Assembly-interface routines

The following routines are not directly callable from "C". A brief summary of the function is given below.

In each case:
ENTRY:    
RR4 = floating point argument 1
RR2 = floating point argument 2 (optional)

EXIT:     
RR2 = resulting number (may be floating point longword integer)

fp_add add RR2 = RR4+RR2 (fp re
fp_mult multiply RR2 = RR2*RR4 (fp re
fp_div divide RR2 = RR2/RR4 (fp re
fp_frac fraction RR2 = frac(RR2) (fp re
fp_cmp compare RR2 = sgn(RR2-RR4) (long
fp_int integer RR2 = int(RR2) (long

Assembly-utility routines

The following routines are not directly callable from "C". A brief summary of the function is given below.

split separates the "sign","exponent", and "mantissa
mul_mant multiplies two mantissas together
div_mant divides two mantissas
unspli recombines the "sign","exponent", and "mantiss

Externals ( out_dec,out_char,in_string,out_crlf,mess_c,out_int )
fpsubs MODULE
EXTERNAL
_out_dec procedure
_out_char procedure
_in_string procedure
_out_crlf procedure
_mess_c procedure
_out_int procedure
GLOBAL
_fadd PROCEDURE
ENTRY
! C callable procedure to perform floating point addition of !
! two numbers (must be floating point) and return the result !
! ENTRY:
! rr6 = addend
! rr4 = additive
! EXIT:
! rr2 = addend + additive
ldl rr2,rr6 ! substitute rr2 for rr6 !
call fp_add ! which is used in fp_add !
ret ! already in rr2 !
END _fadd

_fmul PROCEDURE
ENTRY
! C callable procedure to perform floating point multiplication !
! of two numbers (must be floating point) and return the result !
ENTRY:
! rr6 = multiplicand
! rr4 = multiplier
EXIT:
! rr2 = multiplicand * multiplier
ldl rr2,rr6 ! rr2 for fp_mult !
call fp_mult
ret
END _fmul
_fdiv PROCEDURE
ENTRY
! C callable procedure to perform floating point division!
! of two numbers (must be floating point) and return the result.

ENTRY:

rr6 = dividend
rr4 = divisor

EXIT:
rr2 = dividend/divisor

ldl rr2,rr6
call fp_div
ret
END _fdiv

_fcmp PROCEDURE
ENTRY

! C callable procedure to compare two floating point numbers!
! A and B and return either a -1, 0, or 1 to represent:
! A>B, A=B, and A>B, respectively

ENTRY:

rr6 = A
rr4 = B

EXIT:
rr2 = -1,0,1

ldl rr2,rr6
call fp_cmp
ret
END _fcmp
_fint PROCEDURE
ENTRY
! C callable routine to return a LONG INTeger representing
! the integer part of a floating point number!

ENTRY:
   rr6 = floating point number of return integer part of
EXIT:
   rr2 = integer part of rr6 (not in floating point)

ldl  rr2,rr6
call  fp_int
ret

END _fint

_frac PROCEDURE
ENTRY
! C callable procedure to return the fractional part of a
! floating point number as a floating point number!

ENTRY:
   rr6 = floating point number (input)
EXIT:
   rr2 = fractional part of input (floating point format)

ldl  rr2,rr6
call  fp_frac
ret

END _frac
_fsub PROCEDURE
ENTRY
! C callable procedure to perform subtraction on two floating
! point numbers!
! ENTRY:
rr6 = subtrahend
rr4 = subtractor
EXIT:
rr2 = subtrahend - subtractor
xor r4,%$8000 ! make subtractor negative!
ldl rr2,rr6
call fp_add ! and add to negated subtractor
ret ! the end!
END _fsub

fp_add PROCEDURE
ENTRY
! rr2 contains bit encoded data to add to rr4!
! this routine destroys all other register data and returns the
result in rr2!
ldl arg1,rr2 ! save addend!
ldl arg2,rr4 ! save additive!
cpl rr2,0 ! if addend is zero, then!
jr nz,fpad1 ! return additive, else jump!
ldl rr2,rr4 ! result is additive if zero!
ret ! done!

fpad1:
cpl rr4,0 ! if additive is zero, then!
jr nz,fpad2 ! return addend, else jump!
ret ! additive was zero, so result
! is already in rr2!

fpad2:
call split

testl arg1 ! test for negative mant!
jr pl,fpad3 ! go if positive!
ldl rr6,0 ! make negative by subtraction
subl rr6,rr10 ! and move result back into rr
ldl rr10,rr6 ! done!
fpad3: testl arg2   ! same as for addend!
       jr pl,fpad4
       ldl rrl,#0
       subl rrl,rr8
       ldl rr8,rr6

fpad4: cp rrl,rl2   ! while (aexp <> bexp) ...
       jr z,fpad7   ! go if equal!
       jr gt,fpad5  ! go if aexp > bexp!
       inc rrl,#1   ! increment aexp!
       sral rrl0,#1 ! divide amant by 2!
       jr fpad6    ! skip next part!

fpad5: inc rrl3,#1 ! increment bexp!
       sral rrl8,#1 ! bexp = bexp/2!

fpad6: jr fpad4  ! end while....!

fpad7: cpl rrl0,#0 ! if one was shifted out!
       jr nz,fpad8 ! go if amant is not zero!
       ldl rrl2,arg2
       ret

fpad8: cpl rrl8,#0 ! see if other shifted out!
       jr nz,fpad9
       ldl rrl2,arg1
       ret

fpad9: addl rrl8,rr10  ! add mantissas!
       ! now, resulting mantissa is in rrl8, resulting exponent in rrl3
       ld sign,#0 ! zero negative sign flag!

       testl rrl8  ! see if mantissa negative!
       jr pl,fpad10 ! go if positive!
       ldl rrl,#0 ! else make positive!
       subl rrl,rr8 ! and put back into rrl8!
       ldl rrl,rr6 ! done!
       ld sign,#ff ! set negative sign flag!
       ld rrl2,sign
       call unsplit

       ret

END fp_add
split PROCEDURE
ENTRY
! rr2 is arg 1, rr4 is arg 2!
! results: r12 = exp(arg 1) from RR2
     r13 = exp(arg 2) from RR4
     rr10 = mant(arg 1) from RR2
     rr8 = mant(arg 2) from RR4
     arg1 = rr2
     arg2 = rr4!
     and r2, #7f00          ! mask out exponent of arg1!
exb r12, r2              ! place into lower half!
sub r2, #32              ! subtract off bias!
   ld r12, r2             ! save in r12!

     and r4, #7f00          ! mask out exponent of arg2!
exb r14, r4              ! place into lower half!
sub r4, #32              ! subtract off bias!
   ld r13, r4             ! save in r13!

     ld1 rr2, arg1           ! get uncorrupted arg1 back!
     and r2, #ff             ! only need lower 8 bits!
or r2, #100               ! set hidden bit!
ld1 rr10, rr2             ! save arg2 mant in rr10!

     ld1 rr4, arg2           ! same as above!
     and r4, #ff             !
or r4, #100               !
ld1 rr8, rr4              ! save additive mant in rr8!

ret
END split
GLOBAL

fp_mult PROCEDURE

ENTRY

! rr2 contains bit encoded data to multiply!
! by rr4. This routine destroys all other register!
! data and returns the answer in rr2.!

    ldl arg1,rr2 ! save multiplicand!
    ldl arg2,rr4 ! save multiplier!
    cpl rr2,#0 ! if zero then!
    jr nz,fpmull ! return zero, else jump!
    ldl rr2,#0 ! result is zero!
    ret ! done!

fpmull:
    cpl rr4,#0 ! if zero then!
    jr nz,fpmul2 ! return zero else jump!
    ldl rr2,#0 ! result is zero!
    ret ! done!

fpmul2:
    call split ! mask out exp' and mant'
! r13 is exp and rr10 is mant of multiplicand!
! r12 is exp and rr8 is mant of multiplier!
    cpl rr8,#0 ! if zero then!
    jr nz,fpmul3 ! return zero, else jump!
    ldl rr2,#0 ! result is zero!
    ret ! done!

fpmul3:
    cpl rr10,#0 ! if zero then!
    jr nz,fpmul4 ! return zero, else jump!
    ldl rr2,#0 ! result is zero!
    ret ! done!

fpmul4:
    ldl rr4,rr8 ! set up to multiply the!
    ldl rr6,rr10 ! mantissi together!
    call _mul_mant ! returns result in rr2!
    ld sign,#0 ! start test for sign!
    testl arg1 ! check to see if!
    jr lt,fpmul5 ! result is positive or!
    testl arg2 ! and set sign bit!
    jr lt,fpmul6 ! accordingly!

A-34
jr  fpmul7 ! positive!
testl arg2
jr  gt,fpmul6
jr  fpmul7 ! positive!
fpmul6:
ld  sign,#%ff ! negative!
    ! setup to call unstrip!
    ! resulting mantissa in rr8!
    ! sum of exp in r13!
    ! sign in r12, 0 if pos, ff if
    ! reforms the product into:
    ! a 32 bit fp number!
    ! done!
    ret

END fp_mul

GLOBAL
_mul_mant PROCEDURE
ENTRY
    slal  rr6,#6
    slal  rr4,#6
    ldl  rr2,rr4
    multl  rq4,rr2
    sral  rr4,#4
    ldl  rr2,rr4
    ret
END _mul_mant

_div_mant PROCEDURE
ENTRY
    ldl  rr2,rr4
    ldl  rr4,rr6
    ldl  rr6,#0
    divl  rq4,rr2
    ldl  rr2,rr6
    srl1  rr2,#8
    ret
END _div_mant

A-35
GLOBAL

unspli t PROCEDURE

ENTRY

! enter this routine with cmantissa in rr8 and cexp in r13 !
! sign in r12 , 0 if pos and ff if neg !
! returns result in rr2 !

fpad10: 
  ld  r0,#0
  ld1 rr2,#0
  testl rr8
  ret  z

fpad11: 
  ld1 rr2,rr8
  and r2,#%ff00
  clr  r3
  cpl rr2,#%1000000
  jr  z,fpad14
  cp  r0,#32
  jr  z,fpad14
  cpl rr8,#%1000000
  jr  lt,fpad12
  sral rr8,#1
  inc r13
  jr  fpad13

fpad12: 
  slal rr8,#1
  dec r13
  jr  fpad11

fpad13: 
  inc r0
  jr  fpad11

fpad14: 
  cp  r13,#-31
  jr  gt,fpad15
  ldl rr2,#0
  ret

fpad15: 
  cp  r13,#31
  jr  lt,fpad17
  ldl rr2,#%e fffff
  cp  r12,#%ff
  jr  nz,fpad16
  or  r8,#%8000
  jr  fpad16
  ldl rr2,rr8
  ret

fpad17: 
  ldl rr2,rr8
and r2,$00ff

! clear exponent part!

cmp r12,$fff

! see if negative!

jr nz,fpad18

! go if positive!

or r2,$8000

! or in sign bit!

fpad18:

add r13,$32

! add bias to exponent!

ld r4,r13

ld rh4,r14

or r2,r4

! now, word is complete!

ret

END unsplit

GLOBAL

fp_div PROCEDURE

ENTRY

! rr2 contains bit encoded data to divide by rr4. !
! this routine destroys all other register data !
! and returns the answer in rr2 . !

ex r3,r5

ex r2,r4

ldl arg1,rr2

! save dividend !

ldl arg2,rr4

! save divisor !

cpl rr4,$0

! if zero then, !

jr nz,fpdiv1

! result is zero !

ret ! done !

fpdiv1:

cpl rr2,$0

! if zero then, !

jr nz,fpdiv2

! return max, else jump !

ldl rr2,$3effffff

! result is max !

ret ! done !

fpdiv2:

call split

! mask out exp' and mant' !

! r13 is exp and rr10 is mant of divisor !
! r12 is exp and rr8 is mant of dividend !

cpl rr10,$0

! if zero then !

jr nz,fpdiv3

! return max, else jump !

ldl rr2,$3effffff

! result is max !

ret e !pdi cpl rr8,$0

! r1 is exp and rr1 is mant of dividend !

jr nz,fpdiv4

! return zero, else jump !

ret
ldl rr2,$0
ret
! result is zero !
! done !

fpdiv4:
cpl rr8,rr10
jr lt,fpdiv5
sral rr8,$1
inc r13,$1
jr fpdiv4
! for mant div !
! if dividend => divisor then, !
! divide dividend mant by 2 !
! bump up the exp !
! try till true !

fpdiv5:
ldl rr6,rr8
ldl rr4,rr10
call _div_mant
! set up to divide the mant !
! by one another !
! returns result in rr2 !

fpdiv6:
testl arg1
jr lt,fpdiv6
! check to see if !
! result is positive !

fpdiv7:
ld sign,$0
! start test for sign !

fpdiv8:
testl arg2
jr gt,fpdiv7
jr fpdiv8
! or neg and set sign flag !
! accordingly !

fpdiv8:
ldl rr8,rr2
sub r13,r12
nop
ld r12,sign
call unsplit
! set up for unsplit !
! subtract exponents !
! still setting up for unsplit !
! result in rr2 !

END fp_div
fp_int PROCEDURE
ENTRY
! rr2 contains number to return integer part of in rr2!
ldl argl,rr2 ! save off argument!
call split ! resulting mantissa in rr10,
ld sign, #0 ! determine the sign!
testl argl
jr pl,f pint1 ! go if positive!
ld sign, #ff ! else, flag as negative!
f pint1:
cp r12, #0 ! if exp < 0, then int(arg1)=0
jr ge,f pint2 ! go if exp > 0!
ldl rr2, #0 ! else, result is zero!
ret
f pint2:
cp r12, #31 ! if exp > 31 then overflow!
jr le,f pint3 ! go if no overflow!
ldl rr2, #7fffffff ! rr2 gets max positive value
cp sign, #ff ! see if sign < 0!
ret nz ! done if positive!
addl rr2, #1 ! else make max negative %8000 ! and return!
ret
f pint3:
ldl rr2, #0 ! zero accumulator!
f pint4:
cp r12, #0 ! while exp >= 0!
jr lt,f pint6 ! go if exp < 0!
addl rr2, rr2 ! double accum!
bitt r10, #8 ! check bit of mant!
jr z,f pint5 ! go if bit is zero!
addl rr2, #1 ! increment accum!
f pint5:
and r10, #ff ! clear bit!
addl rr10, rr10 ! double mant!
dec r12, #1 ! decrement exp!
jr f pint4 ! loop while...!
f pint6:
cp . sign, #ff ! if sign <> 0!
jr nz, f pint7 ! then, negate result!
ldl rr8, #0 ! use rr8!
subl rr8, rr2 ! make neg!
ex r2, r8 ! now result in rr2!
ex r3, r9 !
f pint7:
ret ! end of fp_int!
END fp_int
fp_frac PROCEDURE
ENTRY
  ! rr2 is fp number input, rr2 is fp fractional output!
  ldl arg1,rr2
  testl rr2
  jr nz,fpfrcl
  ldl rr2,#0
  ret
fpfrcl:
  call split
  cp rl2,#-1
  jr gt,fpfrc2
  ldl rl2,arg1
  ret
fpfrc2:
  cp rl2,#-1
  jr le,fpfrc3
  dec rl2,#1
  addl rr10,rr10
  jr fpfrc2
fpfrc3:
  and r10,#01ff
  ld sign,#0
  cp arg1,#0
  jr ge,fpfrc4
  ld sign,#ff
  fpfrc4:
  ldl rr8,rr10
  ld rl3,rl2
  ld rl2,sign
  call unsplit
  ret
END fp_frac
_fp_out PROCEDURE
ENTRY
  ld1   rr2,rr6
  ! rr2 has fp number of output !
  ld1   argl,rr2
  call  split
  ! copy C variable to rr2 !
  ld1   argl,rr2
  ! save off !
  call  split
  ! rr10 has mant, r12 has exp !
  test1 argl
  jr    ge,fpout1
  ldb   r17,#%2d
  ! go if >= 0 !
  call  _out_char
  ! assembly routine !
  ld1   rr2,argl
  res   r2,#15
  ld1   argl,rr2
  ! get rr2 back and clear sign
  ! clear the sign bit !
  ! back in argl otay !
fpout1:
  ld1   rrl,rr2
  ! get argl back :!
  call  fp_int
  ! get integer part back !
  pushl @r15,rr2
  ! save on stack !
  ld1   rr2,argl
  ! get argl back !
  call  fp_frac
  ! get fractional part back !
  popl  rrl,rr2
  ! save rr2 !
  call  _out_dec
  ! 'C' routine to output integer
  ldb   R17,#%2e
  ! '!' !
  call  _out_char
  ! output it to CRT !
ipopl  rr2,@r15
  ! get rr2 back !
  ld    dig_count,#0
  ! digit counter !
fpout2:
  test1 rr2
  jr    z,fpout3
  cp    dig_count,#05
  ! stop after 6 digits !
  jr    gt,fpout3
  ! stop if 6 done already !
  inc   dig_count
  ld1   rr4,#%23400000
  ! the number 10 !
  call  fp_mult
  ! result in rr2 !
pushl @r15,rr2
  ! save result of mult !
call  fp_int
  ! get integer part (0-9) !
  ldb   r17,r13
  ! output as digit !
  addb  r17,#%30
  ! ascii offset !
call  _out_char
  ! output to screen !
popl  rr2,@r15
call  fp_frac
  ! get rr2 back !
call  fp_out2
  ! result in rr2 !
call  _out_char
  ! loop till done !
fpout3:
  ret
END _fp_out
dig_count:
    wval  0

fp_cmp PROCEDURE
ENTRY
    ! rr2 is compared with rr4, returns -1,0,1 for <,=,> !
    ldl  arg1,rr2
    ldl  arg2,rr4
    call split
    ldl  rr2,arg1
    ldl  rr4,arg2
    and r2,%8000
    and r4,%8000
    sub r2,r4
    jr eq,fpcmpl
    jr gt,fpcmp3
    jr fpcmp2
fpcmpl:
    cp r12,r13
    jr eq,fpcmp4
    jr gt,fpcmp6
    jr fpcmp7
fpcmp2:
    ldl  rr2,#-1
    ret
fpcmp3:
    ldl  rr2,#1
    ret
fpcmp4:
    subl rr10,rr8
    jr eq,fpcmp5
    jr gt,fpcmp6
    jr fpcmp7
fpcmp5:
    ldl  rr2,#0
    ret
fpcmp6:
    ldl  rr2,#1
    jr fpcmp8
fpcmp7:
    ldl  rr2,#-1
fpcmp8:
    testl arg1
    ret gt
    testl rr2
    ret z
    jr gt,fpcmp2
    jr fpcmp3
END fp_cmp
_fp_in PROCEDURE
ENTRY
! gets characters into array, converts to real in rr2!
ld r7,#string       ! address of string storage!
ld r6,#20           ! num characters!
call _in_string      ! get chars!

fp_in_con:
ld mult_factor,#%2000 ! 1.0!
ld mult_factor+2,#%0000
ld point,#0          ! dec point not encountered!
ld ntor,#0           ! number of digits to right of
ld rr2,#0            ! zero result!
ld r14,#string       ! string pointer!
ld in_sign,#0        ! sign is positive!

fpin1:
testb @r14           ! while still more chars!
jp z,fpin7           ! go if done!
ldb r10,@r14         ! get next character!
cpb r10,#%2e         ! see if '.'!
jp nz,fpin2          ! go if not!
ld point,#1           ! else flag point!

fpin2:
cpb r10,'#-'
jp nz,fpin25         ! see if '-'!
ld in_sign,#1
! else, flag as < 0!

fpin25:
cpb r10,'#E'
jr nz,fpin3          ! see if exponent next!
jp fpin8             ! go if not!
! else, it is, so process!

fpin3:
cpb r10,'#9'
jp gt,fpin45         ! make sure a valid character
! go if too big!
cpb r10,'#0'
jp lt,fpin45
! go if too small!

test point           ! see if to right of '.'!
jp z,fpin4           ! go if not to right yet!

fpin4:
test point
jp nz,fpin40         ! don't mult by 10 if ',' foun
push @r15,r0
ldl rr4,#%234000000  ! constant : 10!
call fp_mult
pop r0,@r15
! get character back!

fpin40:
subb r10,#%30        ! subtract ascii bias!
ldb rh0,#0           ! zero upper half or r0!
add r0,r0            ! double result!
! for indexing into contab!
add r0,r0  ! double again (4 bytes each)
ld r1,r0  ! transfer to r1 for indexing
ldl rr4,contab(r1)  ! get constant !

! now, rr2 is the running result
! rr4 is the digit just typed in
pushl @r15,rr2  ! save rr2 temporarily !

! save running result !
ldl rr2,mult_factor  ! get multiplication factor !
test point  ! if dp found, divide by 10 and
jr z,fpin41  ! go if no point found yet !
pushl @r15,rr4  ! else, save rr4 !
ldl rr4,#234000000  ! divide mult_factor by 10 !
call fp_div  ! do the divide !
ldl mult_factor,rr2  ! new one stored off !
popl rr4,@r15  ! get back digit constant !

ldl rr2,mult_factor  ! get new mult_factor !
fpin41:
call fp_mult  ! multiply digit const by mult
popl rr4,@r15  ! get what was rr2 back (result
call fp_add  ! add result to digit*mult_fac

fpin45:
inc r14  ! bump character pointer !
jp fpin1  ! loop till done !

fpin7:
test in_sign  ! see if < 0 !
ret z  ! finished if not < 0 !
or r2,#8000  ! else, set sign bit !
ret  ! before returning !

fpin8:
! here if 'E' was found in input, indicating exponential notation
pushl @r15,rr2  ! save base10 mantissa on stack
inc r14  ! but first bump char pointer
ld r2,in_sign  ! get in_sign !
ld r3,point  ! and point !
pushl @r15,rr2  ! and save on stack !
ldl rr2,mult_factor  ! get mult_factor !
pushl @r15,rr2  ! and save on stack !
ld in_sign,#0  ! zero out in_sign and point !
ld point,#0  ! so exponent value will be correct
ldl rr2,#200000000  ! back to original mult_factor
ldl mult_factor,rr2  ! zero out accumulator !
ldl rr2,#0
call fpin1
ldl rrr5,rr2
call _fint
ldl rrr4,rr2
popl rrr2,@rl5
ldl mult_factor,rr2
popl rrr2,@rl5
ld in_sign,rr2
ld point,r3
popl rrr2,@rl5

fpin9: testl rrr4
        jr z,fpin7
        jr gt,fpin10
        addl rrr4,#1
        pushl @rl5,rr4
        ldl rrr4,#%23400000
        call fp_div
        popl rrr4,@rl5
        jr fpin9

fpin10:
        subl rrr4,#1
        pushl @rl5,rr4
        ldl rrr4,#%23400000
        call fp_mult
        popl rrr4,@rl5
        jr fpin9

END _fp_in
point:  
ntor:  wval 0  
in_sign:  wval 0  
string:  wval 0  
spaces:  array [25 word]  
        array [*byte] := ' ',  
savel:  array [20 word]  
contab:  
        wval $0000  ! 0.0!  
        wval $0000  
        wval $2000  ! 1.0!  
        wval $0000  
        wval $2100  ! 2.0!  
        wval $0000  
        wval $2180  ! 3.0!  
        wval $0000  
        wval $2200  ! 4.0!  
        wval $0000  
        wval $2240  ! 5.0!  
        wval $0000  
        wval $2280  ! 6.0!  
        wval $0000  
        wval $22c0  ! 7.0!  
        wval $0000  
        wval $2300  ! 8.0!  
        wval $0000  
        wval $2320  ! 9.0!  
        wval $0000  
mult_factor:  
        wval $0000  
        wval $0000
_fpcon PROCEDURE
ENTRY

! C callable procedure to convert a character constant!
! floating point number into an actual floating point number!
! usage:
! fp_number = fpcon("3.14159");

ld r4,#string ! address of string area!
fpcon1:
ldb r10,@r7 ! copy characters into string
! area used by fpin!
inc r7 ! move one!
inc r4
incb r10,#0 ! see if done!
jr nz,fpcon1 ! loop till done (0 found)!
call fp_in_con ! go do it!
ret ! the end!
END_fpcon

.fp_out_e PROCEDURE
ENTRY

! procedure to printout the variable in rr6 in exponential (base 10)

ld exp10,#0 ! base ten exponent is zero fi
ldl rr2,#0
cpl rr6,#0
jr z,fpoe1 ! go if argument is a zero!
ldl templ,rr6 ! save argument!
res r6,#15 ! clear sign bit for compare!
ldl rr4,#20000000 ! constant 1.00!
call _fcmp ! rr2 = -1,0,1 if <1,=1,>1!
testl rr2 ! check sign of comparison!
jp lt,fpoe2 ! go if < 1!
ldl rr6,templ ! see if > 9!
res r6,#15
ldl rr4,#23200000 ! constant 9.00!
call _fcmp ! same return status as above
testl rr2 ! check sign of comparison!
jp gt,fpoe3 ! go if > 9!
ldl rr2,templ
fpoel:   
  ld1 rr6,rr2     ! print mantissa part !
  call _fp_out   ! print 'E' !
  ldb rl7,'#E'   
  call _out_char 
  ld r7,exp10    ! load exp10 into rr6 !
  exts rr6       ! extend sign !
  test rl6       ! see if >= 0 !
  jr lt,fpoel1   ! go if < 0 !
  pushl @r15,rr6  
  ldb rl7,'#+'    ! else, print '+' !
  call _out_char  
  popl rr6,@r15  

fpoel1:  
  call _out_dec  ! output the exponent !
  ld1 rr2,#1     ! return 1 !
  ret

fpoe2:    ! rr7 < 1.0 !
  ld1 rr2,templ  ! get argument !

fpoe22:   
  pushl @r15,rr2  ! save result !
  ld1 rr6,rr2    ! compare with 1.0 again !
  res rr6,#15    ! clear sign bit !
  ld1 rr4,#%20000000  ! 1.0 constant !
  call _fcmp     ! check sign of result !
  testl rr2      ! get result back !
  popl rr2,@r15  ! already ready for output !
  ld1 rr4,#%23400000  ! multiply by 10 !
  call _fp_mult   ! and decrement exponent !
  dec exp10      ! loop till done !
  jr fpoe22

fpoe3:    ! rr7 > 9.0 !
  ld1 rr2,templ  ! save result !

fpoe33:   
  pushl @r15,rr2  ! 9.0 constant !
  ld1 rr6,rr2    
  res rr6,#15    
  ld1 rr4,#%232000000  
  call _fcmp     
  testl rr2      
  popl rr2,@r15  
  jp le,fpoel1   
  ld1 rr4,#%23400000  ! divide by 10.0 !
  call _fp_div    
  inc exp10      
  jr fpoe33      ! loop till ready to print !

END_fp_out_  
templ:       wval $0000
             wval $0000

A-48
exp10: wval $0000
wval $0000

_fpconv PROCEDURE
ENTRY
    ! converts long integer in rr6 to fp number in rr2!
    ldl rr8,rr6
    ld r12,#0
    test1 rr8
    jr ge,fpconv1
    ldl rr2,#0
    subl rr2,rr8
    ldl rr8,rr2
    ld r12,#ff
    fpconv1:
    ld r13,#24
    jp unsplit
END _fpconv

END fp_subs

A-49/(A-50 Blank)
/* program to input and display data from the A/D board */
/* external references */

extern int in_char(), in_int(), wait_char(), out_int(), mess_c();
extern int ad_in_a(), ad_in_b();
int ch_array[8], count2;
int adin[30], *pointer;

/* local variables */
int num_chans, count;
char con_dis, cntrl1;

main()
{
    out_clrf();
mess_c("A/D input and display program ");
    out_clrf();
    out_clrf();
mess_c("Display how many channels (0-7) ");
    num_chans = in_int();
    if (num_chans != 0) {
      if (num_chans > 7) num_chans = 7;
      out_clrf();
mess_c("Number of channels set to: ");
      out_int(num_chans);
      out_clrf();
      for (count-1; count <= num_chans; count++) {
        mess_c("column");
        out_int(count);
mess_c(" ");
        ch_array[count] = in_int();
        out_clrf();
      } /* for */
    out_clrf();
mess_c("Continuous or Discrete sampling (C/D) ");
    con_dis = wait_char();
    if (((con_dis != 'C') & (con_dis != 'D')) con_dis = 'D';
    out_clrf();
mess_c("Sampling set to: ");
    out_char(con_dis);
    out_clrf();
    /* the following is the main routine */

    c = count2 = 0;
    while (((cntrl-in_char()) != 0X1B & c != 0X1B)) {
      if ((cntrl=='C') | (c=='C')) con_dis = 'C';
      else if ((cntrl=='D') | (c=='D')) con_dis = 'D';
      else if (count2 >= 15) /* redisplay channel numbers */
        out_clrf();
      for (count=1; count <= num_chans; count++) {
        mess_c("chan: ");
        out_int(ch_array[count]);
mess_c(" ");
      } /* for */
    }
}
out_clrf();
for (count=1; count<=num_chans; count++)
    mess_c("---------");
out_clrf();
out_clrf();
count2=0;
} /* if */
count2++;
pointer = adin;
ad_in_a(pointer,12,1);
pointer=adin+12;
ad_in_b(pointer,12,1);
for (count=1; count<=num_chans; count++) {
    out_int(adin[ch_array[count]]);
    mess_c(" ");
} /* for */
out_clrf();
if ((cntrl=='P') || (con_dis == 'D')) {
    mess_c("--PAUSED--");
c=wait_char();
out_clrf();
} /* if */
} /* while */
/* main */
}
APPENDIX C
DARAMP

C-1/(C-2 Blank)
/*
** Routine to output a ramp to the D/A's
*/

extern int in_char(), out_crlf(), in_int(), mess_c(), da_out();
extern int cio_unit();
int start_chan, stop_chan, i, chan, num_chans;
int ch_array[20][j, data_array[1], *k;
int start_count, stop_count, delta_count;
char c, d;

main()
{
    cio_unit();
    out_crlf();
mess_c("D/A ramp program ");
    out_crlf();

    mess_c("Input start count ");
    start_count=in_int();
    out_crlf();

    mess_c("Input stop count ");
    stop_count=in_int();
    out_crlf();

    mess_c("Input delta count ");
    delta_count=in_int();
    out_crlf();

    /* get the channel definitions */

    mess_c("Output to how many channels ? (1-10 hex) ");
    num_chans=in_int();

    if (num_chans > 16)
        num_chans=16;

    if (num_chans < 1)
        num_chans=1;
    out_crlf();

    for(j=0; j<(num_chans-1); j++)
    {
        mess_c("Enter channel number ");
        ch_array[j]=in_int();
        out_crlf();
    }

    for(j=0; j<15; j++)
    {
        data_array[0]=0;
        da_out(data_array, j, 1);
    }
/* main routine follows */

mess_c("starting ramp ");
out_crlf();

while( ((c=in_char())!=0x1b) & (d!=0x1b) )
{
    for(i=start_count; i<stop_count; i+=delta_count)
    {
        for(j=0; j<(num_chans-1); j++)
        {
            da_out(data_array, ch_array[j],1);
            data_array[0]=1;
            d=in_char();
        }
    }
}

} /* MAIN() */
APPENDIX D

ADTODA

D-1/(D-2 Blank)
/*  ad_co_da.c :  
**    program to routine an A//D channel  
**       to a D//A channel  
*/

extern int da_out(), ad_in(), mess_c(), in_char(), out_crlf(), in_int();
extern int out_int(), out_char();
int da-chan, ad-chan, adin[30];
char c, d;

main()
{
    out_crlf();
    mess_c("A//D to D//A routing program ");
    out_crlf();
    out_crlf();

    mess_c("A//D channel number ");
    ad-chan = in_int();
    out_crlf();

    mess_c("D//A channel number ");
    da-chan = in_int();
    out_crlf();

    if ( da-chan>15 )
        da-chan = 15;

    mess_c("Routing started ");
    out_crlf();
    out_crlf();

    while( (c=in_char())!=0x1b )
    {
        ad_in(adin);
        da_out(da_chan, adin[ad-chan]);
    }

    out_crlf();
    out_crlf();

} /* MAIN() */
APPENDIX E

DSCOUT

E-1/(E-2 Blank)
/* Program to test the discrete output */

extern int in_char(), out_int(), mess_c(), out_crlf();
extern int cio_unit(), cio_outa();
int b, i;
char c;

main()
{

cio_unit();
mess_c("Discrete output test routine ");
out_crlf();
out_crlf();
mess_c("Starting sequence ");
out_crlf();
while((c=in_char())!=0x1b)
{
    b = 1;
    while((b<=128) & ((c=in_char())!=0x1b))
    {
        cio_outa(b);
        b *= 2;
        for(i=0; i<32766; i++)
            ; /* delay */
    }
    /* now blink and invert */
    b = 1;
    while((b<=128) & ((c=in_char())!=0x1b))
    {
        i = b^0xff;
        cio_outa(i);
        b *= 2;
        for(i=0; i<32766; i++)
            ; /* delay */
    }
}

} /* main() */
APPENDIX F

ACTIVITY

F-1/(F-2 Blank)
/* routine to check out the activity circuitry */

extern int cio_init(),cio_outb(),cio_outc(),mess_c(),out_crlf();

int i,j,k,data[3],*point;
char c,d,e;

main()
{
    cio_init();
    out_crlf();
    mess_c("Starting activity test...");
    out_crlf();
    for (j=0; j<=3; j++)
    {
        mess_c("D/A channel ");
        out_int(j);
        mess_c(" test");
        out_crlf();
        for (i=0; i<=16384; i++)
        {
            data[0]=0x4000;
            point= data;
            da_out(point,j,1); /* address=point, channel=j, data's=1 */
        }
    }
    out_crlf();
    mess_c("All channels ");
    out_crlf();

    while (1)
    {
        for (i=0; i <= 32766; i++)
        {
            c = in_char();
            j = 0;
            point = data;
            da_out(point,j,4);
            if( c==0x1b )
                break;
        }
        if( c==0x1b )
            break;
    }

    /* reset all the activity lights */
}

/* main */

F-3/(F-4 Blank)
# DISTRIBUTION

<table>
<thead>
<tr>
<th>University of Alabama in Huntsville Off Campus Research Facility</th>
<th>No. of Copies</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTN: AMSMI-RD-SS-SE, Bldg 5400</td>
<td>10</td>
</tr>
</tbody>
</table>

| US Army Materiel System Analysis Activity                       |               |
| ATTN: AMXSY-MP                                                 | 1             |
| Aberdeen Proving Ground, MD 21005                               |               |

| ITT Research Institute                                         |               |
| ATTN: GACIAC                                                   | 1             |
| 10 W. 35th Street                                              |               |
| Chicago, IL 60616                                              |               |

| AMSMI-RD, Dr. McCorkle                                         | 1             |
| Dr. Rhoades                                                    | 1             |
| AMSMI-RD-CS-R                                                  | 15            |
| AMSMI-RD-CS-T                                                  | 1             |
| AMSMI-RD-SS-SE, Dr. Hallum                                    | 1             |
| AMSMI-GC-IP, Mr. Bush                                          | 1             |

DIST-1/(DIST-2 Blank)
END
6 - 87
DTIC