IDENTIFICATION

PRODUCT CODE: MAINDEC-11-DZRK1-E-D
PRODUCT NAME: RK11 BASIC LOGIC TEST I
DATE CREATED: APRIL, 1977
MAINTAINER: DIAGNOSTIC GROUP
AUTHOR: JIM KAPADIA
REVISED BY: PERVEZ ZAKI
           TOM SAWYER
           CHUCK HESS

THE INFORMATION IN THIS DOCUMENT IS SUBJECT TO
CHANGE WITHOUT NOTICE AND SHOULD NOT BE CONSTRUED AS
A COMMITMENT BY DIGITAL EQUIPMENT CORPORATION.
DIGITAL EQUIPMENT CORPORATION ASSUMES NO
RESPONSIBILITY FOR ANY ERRORS THAT MAY APPEAR IN
THIS MANUAL.

THE SOFTWARE DESCRIBED IN THIS DOCUMENT IS FURNISHED
TO THE PURCHASER UNDER A LICENSE FOR USE ON A SINGLE
COMPUTER SYSTEM AND CAN BE COPIED (WITH INCLUSION OF
DIGITAL'S COPYRIGHT NOTICE) ONLY FOR USE IN SUCH
SYSTEM EXCEPT AS MAY OTHERWISE BE PROVIDED IN
WRITING BY DIGITAL.

DIGITAL EQUIPMENT CORPORATION ASSUMES NO
RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
SOFTWARE ON EQUIPMENT THAT IS NOT SUPPLIED BY
DIGITAL.

COPYRIGHT (C) 1975, 1977 BY DIGITAL EQUIPMENT CORPORATION
QUICK LOOK-UP OPERATING INSTRUCTIONS

FOR A QUICK REFERENCE, LOOK UP THE FOLLOWING SECTIONS:

1.0 ABSTRACT
2.0 REQUIREMENTS
4.1 LOADING AND OPERATOR ACTION
5.0 SWITCH OPTIONS

FOR A MORE COMPLETE EXPLANATION REFER TO THE TABLE OF CONTENTS BELOW AND THE FOLLOWING DOCUMENT.

TABLE OF CONTENTS

1.0 ABSTRACT

2.0 REQUIREMENTS
   EQUIPMENT
   PRELIMINARY PROGRAMS
   EXECUTION TIME

3.0 STARTING ADDRESS

4.0 PROGRAM CONTROL MODES & OPERATOR ACTION
   PAPER TAPE
   RKDP DUMP MODE
   RKDP CHAIN MODE
   ACTII

5.0 SWITCH OPTIONS

6.0 SCOPE LOOPS

7.0 PROGRAM STRUCTURE

8.0 ERROR REPORTING

9.0 ERROR INTERPRETATION

10.0 HANDLERS AND COMMON ROUTINES
    TRAP HANDLER
    SCOPE HANDLER
    ERROR HANDLER

10.1 CONTROL RESET ROUTINE
    CONTROL READY ROUTINE
    TIME DELAY ROUTINE
    OTHER ROUTINES
    TTY HANDLER (I/O), ERROR TYPEOUT ROUTINE
    POWER DOWN/POWER UP ROUTINE

11.0 UNEXPECTED TIMEOUTS & RK11 INTERRUPTS

15.0 QUICK VERIFYING MODE
1.0 ABSTRACT

THE RK11 LOGIC TESTS CONSIST OF A SERIES OF TESTS AIMED AT CHECKING THE BASIC LOGIC OF THE RK11 CONTROLLER.

THE LOGIC TESTS CONSIST OF TWO PARTS. THIS PROGRAM IS PART-I AND IT CHECKS ONLY THE DRIVE-INDEPENDENT LOGIC OF THE RK11 CONTROLLER (SEE SEC. 9-O). IT SHOULD BE NOTED THAT LOGIC TEST-I AND LOGIC TEST-II TOGETHER CONSTITUTE A COMPLETE PROGRAM AND HENCE BOTH OF THEM SHOULD BE RUN.

USED CORRECTLY THIS PROGRAM CAN BE AN EFFECTIVE ANALYTIC AND DIAGNOSTIC TOOL.

2.0 REQUIREMENTS

2.1 EQUIPMENT
A. PDP11 WITH CONSOLE TELETYPE.
B. BK OF MEMORY
C. RK11 OR RKVII CONTROLLER

2.2 PRELIMINARY PROGRAMS
NONE

2.3 EXECUTION TIME

ERROR FREE FIRST PASS ON PDP11/20 WITH CORE MEMORY TAKES APPROXIMATELY ONE MINUTE. CONSIDERABLY LESS FOR FASTER MACHINES OR MEMORIES.

3.0 STARTING ADDRESS

200 FOR ANY MODE OF OPERATION. NORMAL START UP WITH ALL SWITCHES DOWN.

4.0 PROGRAM CONTROL MODES & OPERATOR ACTION

4.1 PAPER TAPE LOADING

4.1.1 LOAD PROGRAM INTO MEMORY USING STANDARD PROCEDURE FOR .ABS TAPES.
4.1.2 Put the drives on 'WRT PROT' and 'LOAD' as a precaution against malfunctioning.

4.1.3 Load address 200

4.1.4 Set switches if desired (see sec 5.0 if testing on simulator put sw 10 up.

Press start.

4.1.5 The program identifies itself (name, maindec no).

RKII LOGIC TEST I
MAINDEC-11-DZJKJ-E

4.1.6 Then the program proceeds with testing. At the end of a pass the following type-out occurs

END PASS # x

WHERE x = PASS NUMBER (1,2,3---), control is passed to the beginning of the program and re-execution begins.

4.1.7 Error free passes of the program appear as shown below.

RKII LOGIC TEST I
MAINDEC-11-DZJKJ-E
END PASS # 1
END PASS # 2

...

4.2 RKDP dump mode

4.2.1 The program is loaded into the memory by the RKDP monitor

4.2.2 Start as normally using SA 200

4.2.3 The program identifies itself (name, maindec no.) and proceeds with testing.

4.3 RKDP chain mode

The program is chain-loaded from the RKDP pack. After the program identifies itself, it proceeds with testing.

4.4 ACTII mode

The program is loaded by the ACTII monitor. On starting, it proceeds with the execution of the test as before, but the title is not typed out.
5.0 SWITCH OPTIONS

IF THE PROGRAM IS BEING RUN ON A SWITCHLESS PROCESSOR (I.E. AN 11/34),
THE PROGRAM WILL DETERMINE THAT THE HARDWARE SWITCH REGISTER IS
NOT PRESENT AND WILL USE A 'SOFTWARE' SWITCH REGISTER. THE
'SOFTWARE' SWITCH REGISTER IS LOCATED AT LOCATION 176 (8). THE
SETTINGS OF THE 'SOFTWARE' SWITCHES ARE CONTROLLED THROUGH A KEYBOARD
ROUTINE WHICH IS CALLED BY TYPING A 'CONTROL G'. THE PROGRAM WILL
RECOGNIZE THE 'CONTROL G' WHENEVER THE PROGRAM ENTERS
THE SCOPE ROUTINE OR BEGINS A NEW TEST. THE
'SOFTWARE' SWITCH VALUES ARE ENTERED AS AN OCTAL NUMBER IN RESPONSE
TO THE PROMPT FROM THE SWITCH ENTRY ROUTINE:

'SWR = NNNNNN  NEW ='

EACH TIME SWITCH SETTING ARE ENTERED, THE ENTIRE SWITCH REGISTER
IMAGE MUST BE ENTERED. LEADING ZEROS ARE NOT REQUIRED. 'RJBOU' AND
'CONTROL U' FUNCTIONS MAY BE USED TO CORRECT TYPING ERRORS
DURING SWITCH ENTRY.

ON PROCESSORS WITH HARDWARE SWITCH REGISTERS, THE 'SOFTWARE' SWITCH
REGISTER MAY BE USED. IF THE PROGRAM FINDS ALL 16 SWITCHES IN "UP"
'UP' POSITION, ALL SWITCH REGISTER REFERENCES WILL BE TO THE
'SOFTWARE' REGISTER AND THE PROCEDURES DESCRIBED ABOVE MUST
BE FOLLOWED.

SW<15>=1  HALT ON ERROR
SW<14>=1  LOOP ON TEST
SW<13>=1  INHIBIT ERROR PRINTOUTS
SW<12>=1  CYCLE ON ERROR TO THE PREVIOUS
'SCOPE' STATEMENT
SW<11>=1  INHIBIT ITERATIONS
SW<10>=1  TESTING ON SIMULATOR
SW<09>=1  LOOP ON SPECIFIC ERROR
SW<08>=1  LOOP ON TEST AS PER SW<07:00>

5.1 SW<15>

THE PROGRAM HALTS ON ENCOUNTERING AN ERROR AFTER
TYING OUT THE ERROR MESSAGE AND PERTINENT
INFORMATION. PRESSING 'CONTINUE' RESTORES NORMAL
OPERATION OF THE PROGRAM.

5.2 SW<14>

THE PROGRAM LOOPS ON THE SUBTEST THAT IS BEING
EXECUTED WHEN THE SWITCH IS PUT ON. THIS SWITCH IS
USED NORMAL ALONG WITH SW 15. SEE SEC 8.0.

5.3 SW <13>

THIS SWITCH INHIBITS ALL ERROR MESSAGES. NORMALLY
USED WHEN LOOPING ON TEST (SW 14) OR LOPPING ON
5.4 SW (12)

THIS SWITCH ALLOWS THE PROGRAM TO CYCLE FROM THE
POINT OF ERROR TO THE PREVIOUS SCOPE STATEMENT.
NOTE THAT IN DOING SO ANY INITIALIZATION BEING DONE
AT THE BEGINNING OF THE STATEMENT WILL BE DONE AGAIN
AND AGAIN. SEE SEC 8.0 FOR DIFFERENT SCOPE LOOPS
AVAILABLE.

5.5 SW (11)

EACH SUBTEST WILL BE EXECUTED ONLY ONCE. NORMALLY
AFTER THE FIRST PASS, EACH SUBTEST IS ITERATED A
NUMBER OF TIMES (USUALLY 50 TO 5 IN SOME CASES).
SETTING THIS SWITCH INHIBITS ITERATIONS, SO THAT
QUICK PASSES CAN BE MADE.

5.6 SW (10)

THIS SWITCH WHEN SET INDICATES THAT TESTING IS BEING
DONE ON A SIMULATOR. THE SWITCH SHOULD BE PUT UP
BEFORE STARTING THE PROGRAM. NOTE THAT RK11C
IS NOT COMPATIBLE WITH THE SIMULATOR.

5.7 SW (09)

THIS SWITCH PROVIDES THE TIGHTEST POSSIBLE SCOPE
LOOP. NOTE THAT UNLIKE SW12 THE INITIALIZATION OF
PARAMETERS AT THE BEGINNING OF THE SUBTEST MAY NOT
BE DONE IN THIS CASE. THIS SWITCH IS HELPFUL WHEN A
PARTicular PART OF A SUBTEST IS BEING REPEATED USING
DIFFERENT PARAMETERS AND YOU WANT TO SCOPE ON THE
PARAMETER IN ERROR. (EXAMPLE: RKDA IS BEING WRITTEN
AND READ BACK WITH COUNT PATTERNS FROM 1 TO 17777.
PATTERN 561 IS GIVING ERROR, YOU MIGHT NOT WANT TO
GO THROUGH THE 560 PATTERNS BEFORE HITTING ERROR ON
THE 561TH PATTERN. IN THIS CASE SW 9 WILL GIVE YOU
A SCOPE LOOP ON THE 561TH PATTERN ONLY.)

5.8 SW (08)

THIS SWITCH IS USED TO SELECT A PARTICULAR TEST (AS
PER SW (00-07)) FOR EXECUTION AND SUBSEQUENT LOOPING.
THUS IF TEST 15 IS TO BE SELECTED THE SWITCH SETTING
WOULD BE 000015. IT SHOULD BE NOTED THAT BEFORE
SELECTING TEST 15, ALL THE PREVIOUS TESTS (1-14)
WILL BE EXECUTED.

6.0 SCOPE LOOPS

THERE ARE THREE KINDS OF SCOPE LOOPS AVAILABLE
1. SW14: LOOPING IS DONE FOR THE ENTIRE SUB-TEST

2. SW12: LOOPING IS DONE FROM THE POINT OF ERROR BACK TO THE PREVIOUS 'SCOPE' STATEMENT.

3. SWD9: PROVIDE THE TIGHTEST POSSIBLE SCOPE LOOP SEE SEC. 5.7

EXAMPLE:

TST1:  SCOPE

   :   INITIALIZATION
   :   ERROR 1
   :   ERROR 2
   :   ERROR 3
   :   ERROR 4

TST2:  SCOPE

THE SEQUENCE OF LOOPING FOR DIFFERENT CASES IS EXPLAINED BELOW. NOTE THAT 'TST1' AND 'TST2' ARE TAGS WHICH DEFINE THE BOUNDARY OF A TEST. (IN THIS CASE TEST 1) TEST 1 STARTS AT 'TST1' AND ENDS JUST BEFORE 'TST2'.

IN THE ILLUSTRATION BELOW --> INDICATES THE POINT FROM WHERE RETURN IS MADE AND LOOPING IS DONE.

1. ERROR 2 OCCURS, SW 14 SET.
   TST1..ERROR 2..TST2-->TST1..ERROR 2..TST2-->TST1...

2. ERROR 2 OCCURS, SW 12 SET.
   TST1..ERROR 2-->TST1..ERROR2-->TST1...

3. ERROR 2,3; SW 14 SET.
   TST1..ERROR 2..ERROR 3..TST2-->TST1..ERROR 2..ERROR 3..TST2-->TST1...

4. ERROR 2,3; SW 12 SET.
   TST1...ERROR 2-->TST1...ERROR 2-->TST1...

NOTE THAT LOOPING IS DONE FROM THE VERY FIRST ERROR ENCOUNTERED. THE MORE BASIC AND ERROR THE EARLIER IT OCCURS AND IS DETECTED AND SHOULD BE FIXED.

TST1: SCOPE
      : INITIALIZATION
      : ERROR 1

      MOV #1S.$LPERR ; $LPERR CONTAINS
      ; THE ADDRESS TO LCO
      ; BACK ON ERROR- SW 9
      
      I$: ---
      ---

      ERROR 2
      N REPETITIONS
      I

TST2: SCOPE

1. SW 12 SET, ERROR 2 OCCURS DURING K.TH REPETITIONS
TST1..1,2...K.ERROR 2-->TST1..1,2...K.ERROR 2-->TST1..

2. SW 9 SET, ERROR 2 OCCURS DURING K.TH REPETITION
I$.K..ERROR 2-->I$.K..ERROR 2-->I$...

7.0 PROGRAM DESCRIPTION

IN THIS PART OF THE PROGRAM THAT PART OF THE RK11 CONTROLLER IS CHECKED WHICH DOES NOT DEPEND ON SIGNALS FROM THE DRIVE. THUS A DRIVE IS NOT NEEDED FOR THIS TEST, BUT IT SHOULD BE NOTED THAT THE PART-II OF THE 'BASIC LOGIC TESTS' MUST BE RUN, IN ORDER TO GET A COMPLETE COVERAGE.

THE TESTS ARE GRADUALLY BUILT UP, CHECKING THE MOST BASIC AND SIMPLE LOGIC FIRST AND THEN PROGRESSIVELY MORE COMPLEX LOGIC.

THE FIRST TEST CHECKS THAT ALL RK11 REGISTERS CAN BE REFERENCED, WITHOUT TIMING OUT. THEN THE INITIALIZATION LOGIC OF RK11 IS CHECKED, THEN IT IS CHECKED THAT ALL REGISTERS CAN BE WRITTEN AND READ CORRECTLY, BY FLOATING A '1' AND THEN USING A COUNT PATTERN. THEN IT IS CHECKED THAT THE RK11 REGISTERS CAN BE CLEARED USING CONTROL RESET AND.RESET (BUS INIT). FINALLY, THE WORD AND BYTE ADDRESSING LOGIC OF RK11 IS CHECKED TO SEE THAT EACH REGISTER IS UNIQUELY ADDRESSED.
9.0  ERROR REPORTING

The error table starting at $ERRTB contains information pertaining to every error that can occur. Each item in the table consists of four entries.

A. EM - This is a pointer to the error message to be typed out when the error occurs.

B. DH - This is a pointer to the data header to be typed out.

C. DI - This is a pointer to the data which is to be typed out under the headers.

D. O - This is a terminator signifying the end of the item.

The error call is an EMT instruction with its lower byte encoded to indicate the error number. Thus “ERROR 1” would be (EMT+1) 1E 104001.

Every error corresponds to an item in the error table, thus “ERROR 14” would correspond to item 14. As far as possible, the error messages have been kept short, but clarity is not sacrificed for brevity. Instead of this, if the user finds a need he can look up the entire error message in the error items table found in the beginning of the listings. Thus for “ERROR 14”, “ITEM 14” in the item table can be looked up. When the error instruction is executed a trap occurs to the error handler located at $ERROR which processes the error call. See Sec 12.3

9.0  ERROR INTERPRETATION

Whenever an error message is printed out, all registers and other data pertaining to the error are also given. RXDS, RXER, RXBA indicate the contents of the corresponding registers at the time of error.

Every error message contains a PC. This PC indicates the position in program where the error call is located. The error message, because of practical considerations is made short and meaningful. The user is advised to look up the PC in the program listing, where he will find more information about the error. In many instances, a single fault will give rise to more than one error report. A little deliberation and careful
EXAMINATION OF THE DATA GIVEN WILL BE CERTAINLY VERY
HELPFUL IN PINPOINTING THE FAULT. A BRIEF
EXPLANATION OF WHAT IS BEING CHECKED IN THE SUBTEST
IS GIVEN AT THE BEGINNING OF EVERY SUBTEST. ALL THE
NUMBERS GIVEN WITH ERROR MESSAGES ARE IN OCTAL.

10.0 HANDLERS AND COMMON ROUTINES

THE COMMONLY USED ROUTINES USED IN THE PROGRAM ARE
CALLED IN TWO WAYS.
A. AS A SUBROUTINE THROUGH 'JSR' CALL
B. THROUGH A 'TRAP' HANDLER

10.1 TRAP HANDLER

MANY COMMONLY USED ROUTINES IN THE PROGRAM ARE
CALLED USING THE TRAP INSTRUCTION AND THE 'TRAP'
HANDLER. THE LOWER BYTE OF THE TRAP INSTRUCTION IS
ENCODED DIFFERENTLY FOR DIFFERENT ROUTINES. THE
TRAP HANDLER IS LOCATED AT '$TRAP'. WHEN A CALL FOR
A ROUTINE IS EXECUTED, A TRAP OCCURS TO THE HANDLER
AT '$TRAP'. THE HANDLER PICKS UP THE LOWER BYTE OF
THE 'CALL INSTRUCTION' AND USES IT TO FORM THE
STARTING ADDRESS OF THE ROUTINE TO GO TO FOR
SERVICE.

10.2 SCOPE HANDLER

THE 'IOT' TRAP IS USED BY THE 'SCOPE' STATEMENT.
WHEN 'SCOPE' IS EXECUTED, AN IOT TRAP OCCURS TO
MEMORY LOCATION '$SCOPE'. THE SCOPE HANDLER STARTS
AT '$SCOPE'. DEPENDING ON THE SWITCH SETTINGS THE
HANDLER DECIDES TO LOOP ON TEXT, INHIBIT ITERATIONS
ETC. THERE ARE CERTAIN POINTERS AND FLAGS WHICH ARE
ADJUSTED. THEREFORE, IT IS NOT ADVISABLE TO START
THE PROGRAM AT ANY GIVEN LOCATION SINCE THE VARIOUS
POINTERS AND FLAGS MAY NOT BE CORRECTLY ADJUSTED.

10.3 ERROR HANDLER

AN EMT TRAP INSTRUCTION IS USED BY THE ERROR CALL.
THE LOWER BYTE IS ENCODED TO GIVE DIFFERENT ERROR
CALLS. (EX: ERROR 1 = 104000+1; ERROR 16 =
104000+16). WHEN THE ERROR STATEMENT IS EXECUTED, A
TRAP OCCURS TO MEMORY LOCATION '$ERROR'. THE ERROR
HANDLER IS LOCATED AT '$ERROR'. THE HANDLER FORMS
THE POINTER TO ERROR TABLE WHICH IS USED IF AN
ERROR MESSAGE IS TO BE TYPED OUT. DEPENDING ON THE
SWITCH SETTINGS, A DECISION ABOUT HALTING ON ERROR.
INHIBITING TYPEOUT, LOOPING ON ERROR ETC. IS MADE. IF AN ERROR MESSAGE IS TO BE TYPED OUT AN EXIT IS MADE TO THE ERROR MESSAGE TYPEOUT ROUTINE LOCATED AT 'ERRTYP'.

10.4 CONTROL RESET ROUTINE

THE CALL FOR THIS ROUTINE IS “CNT.RESET” AND IS AN ENCODED ‘TRAP’ INSTRUCTION. WHEN “CNT.RESET” IS EXECUTED THE CONTROL RESET ROUTINE STARTING AT “CN.RST” IS ENTERED. A CONTROL RESET IS ISSUED AND THE PROGRAM WAITS TILL THE CONTROL READY SETS, ON WHICH THE ROUTINE IS EXITED. IF CONTROL READY DOES NOT SET WITHIN A CERTAIN TIME AN ERROR IS REPORTED. THE PC TYPED OUT IS THE LOCATION WHERE THE “CNT.RESET” CALL IS LOCATED. THE WAITING TIME IS 2.0 MS FOR 11/20 AND 560 US FOR 11/45 WITH BIPOLAR MEMORY.

10.5 CONTROL READY ROUTINE

THIS ROUTINE IS CALLED BY “CNT.RDY” (AN ENCODED ‘TRAP’ INSTRUCTION) AND IS LOCATED AT “CN.RDY”. THE ROUTINE WAITS FOR THE CONTROL READY TO SET AND WHEN IT DOES, EXITS OUT. IF CONTROL READY DOES NOT SET WITHIN A SPECIFIED TIME AN ERROR MESSAGE IS GIVEN.

CNTRL RDY DIDN’T SET
PC = XXXXXX     RKCS = YYYY

THE PC IS THE LOCATION AT WHICH THE “CNT.RDY” CALL IS LOCATED. THE WAITING TIME IS 949 MS FOR 11/20 AND 189 MS FOR 11/45 WITH BIPOLAR MEMORY.

10.6 TIME DELAY ROUTINE

THIS ROUTINE PROVIDES A VARIABLE TIME DELAY. THE CALL IS DELAY N WHERE N=1 TO 17777 (OCTAL) TIME DELAY PROVIDED= 7.5 TIMES(X) N MICRO SECS FOR 11/20, 15N US FOR 11/45 (N CONVERTED TO DECIMAL BEFORE COMPUTING DELAY) IF THE USER WANTS TO CHANGE THE DELAY AT ANY POINT IT CAN BE DONE BY SIMPLY CHANGING VARIABLE ‘N’.

10.7 OTHER ROUTINES

THERE ARE OTHER COMMONLY USED ROUTINES AS LISTED BELOW.

$TYPE:

TYPE ROUTINE FOR TYPING OUT ASCII STRINGS.
LOCATED AT "$TYPE"
CALLED BY "TYPE"

$TYPOC:
ROUTINE FOR TYPING OUT OCTAL NUMBERS.
LOCATED AT "$TYPOC"
CALLED BY "TYPOC"

$TYPODS:
ROUTINE FOR TYPING OUT DECIMAL NUMBERS.
LOCATED AT "$TYPODS"
CALLED BY "TYPODS"

$SERRTYP:
ROUTINE FOR TYPING OUT ERROR MESSAGES.
LOCATED AT $SERRTYP
CALLED BY "JSR $SERRTYP"

$SPWRDN,$SPWRUP:
ROUTINE FOR HANDLING POWER FAILURE/POWER UP.
LOCATED AT $SPWRDN,$SPWRUP
$SPWRUP CALLED WHEN THERE IS A POWER FAILURE.
$SPWRUP CALLED WHEN THERE IS A POWER UP.

11.0 UNEXPECTED TIMEOUTS AND RK11 INTERRUPTS

WHEN AN UNEXPECTED TIMEOUT OCCURS, THE PC AT WHICH
TIME OUT OCCURRED IS TYPED OUT AND THE PROGRAM HALTS.
IF IT IS INTACT, IT CAN BE RESTARTED BY PRESSING
CONTINUE.

IF AN UNEXPECTED RK11 INTERRUPT OCCURS, THE PROGRAM
TYPES OUT THE PC AT WHICH THE INTERRUPT CAME IN AND
THEN HALTS. PRESSING CONTINUE WOULD RESTART THE
PROGRAM FROM BEGINNING. SW 9-LOOPING CAPABILITY IS
PROVIDED AS A TROUBLESHOOTING AID.

12.0 QUICK VERIFYING MODE

THE FIRST PASS OF THE PROGRAM IS A QUICK VERIFYING
MODE. ALL THE TESTS ARE DONE ONLY ONCE. ON
SUBSEQUENT PASSES THE TESTS ARE ITERATED (NORMALLY
50 TIMES, 5 IN SOME CASES). THEREFORE, THE FIRST PASS
TAKES A SHORTER TIME TO COMPLETE, WHILE SUBSEQUENT
PASSES TAKE MORE TIME.
TITLE MD-11-DIRKJ-E, RKL1 BASIC LOGIC TEST 1
COPYRIGHT (C) 1974, 1977
DIGITAL EQUIPMENT CORP.
MAYNARD, MASS. 01754

PROGRAM BY JIM KAPADIA

* This program was assembled using the PDP-11 MAINDEC SYMEX
* JANUARY 1975
SBTL OPERATIONAL SWITCH SETTINGS

<table>
<thead>
<tr>
<th>SWITCH</th>
<th>USE</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>HALT ON ERROR</td>
</tr>
<tr>
<td>14</td>
<td>LOOP ON TEST</td>
</tr>
<tr>
<td>13</td>
<td>INHIBIT ERROR TYPEOUTS</td>
</tr>
<tr>
<td>12</td>
<td>CYCLE ON ERROR TO PREVIOUS 'SCOPE' STATEMENT</td>
</tr>
<tr>
<td>11</td>
<td>INHIBIT ITERATIONS</td>
</tr>
<tr>
<td>10</td>
<td>TESTING ON SIMULATOR</td>
</tr>
<tr>
<td>9</td>
<td>LOOP ON ERROR</td>
</tr>
<tr>
<td>8</td>
<td>LOOP ON TEST IN SWP(7:0)</td>
</tr>
</tbody>
</table>

* PROGRAM REVISED BY TOM SAWYER, MARCH 1976
* REVISED BY CHUCK HESS, AUGUST 1976

******************************************************************************
YOU ARE ADVISED TO READ THE DOCUMENT BEFORE USING THIS PROGRAM.
ON GETTING AN ERROR REFER TO THE LISTINGS AT THE PC POINTED OUT IN THE ERROR MESSAGE. ADJACENT ERROR MESSAGES IF FOLLOWED CAREFULLY COULD LEAD TO AN EASY PINPOINTING OF THE FAULT

******************************************************************************
SBTL BASIC DEFINITIONS

* INITIAL ADDRESS OF THE STACK POINTER *** 1100 ***
D00100
STACK= 1100
.EQUIV EMX.ERROR : BASIC DEFINITION OF ERROR CALL
.EQUIV IO7,SCOPE : BASIC DEFINITION OF SCOPE CALL

*MISCELLANEOUS DEFINITIONS
D00011
HT= 11 : CODE FOR HORIZONTAL TAB
LF= 12 : CODE FOR LINE FEED
CR= 15 : CODE FOR CARRIAGE RETURN
CRLF= 200 : CODE FOR CARRIAGE RETURN+LINE FEED
PS= 177776 : PROCESSOR STATUS WORD
.EQUIV PS,PSW
BASIC DEFINITIONS

STACK LIMIT REGISTER
STKM= 177774

PROGRAM INTERRUPT REQUEST REGISTER
PIRQ= 177772

HARDWARE SWITCH REGISTER
DSW= 177570

HARDWARE DISPLAY REGISTER
DDISP= 177570

GENERAL PURPOSE REGISTER DEFINITIONS
AC= 0   : GENERAL REGISTER
R0= 1   : GENERAL REGISTER
R1= 2   : GENERAL REGISTER
R2= 3   : GENERAL REGISTER
R3= 4   : GENERAL REGISTER
R4= 5   : GENERAL REGISTER
R5= 6   : GENERAL REGISTER
R6= 7   : GENERAL REGISTER
SP= 8   : STACK POINTER
PC= 9   : PROGRAM COUNTER

PRIORITY LEVEL DEFINITIONS
PR0= 0 : PRIORITY LEVEL 0
PR1= 40 : PRIORITY LEVEL 1
PR2= 80  : PRIORITY LEVEL 2
PR3= 120 : PRIORITY LEVEL 3
PR4= 160 : PRIORITY LEVEL 4
PR5= 200 : PRIORITY LEVEL 5
PR6= 240 : PRIORITY LEVEL 6
PR7= 280 : PRIORITY LEVEL 7

SWITCH REGISTER SWITCH DEFINITIONS
SW15= 100000
SW14= 40000  : EQUIV SW9, SW3
SW13= 20000  : EQUIV SW8, SW4
SW12= 10000  : EQUIV SW7, SW5
SW11= 4000   : EQUIV SW6, SW6
SW10= 2000   : EQUIV SW5, SW5
SW9= 1000   : EQUIV SW4, SW7
SW8= 400    : EQUIV SW3, SW8
SW7= 200    : EQUIV SW2, SW9
SW6= 100    : EQUIV SW1, SW10
SW5= 40     : EQUIV SW0, SW11
SW4= 20     : EQUIV SW12, SW12
SW3= 10     : EQUIV SW13, SW13
SW2= 4      : EQUIV SW14, SW14
SW1= 2      : EQUIV SW15, SW15
SW0= 1      : EQUIV SW16, SW16

DATA BIT DEFINITIONS BIT00 TO BIT15
```
000000  BIT15= 100000
000000  BIT14= 400000
000000  BIT13= 200000
000000  BIT12= 100000
000000  BIT11= 400000
000000  BIT10= 200000
000000  BIT09= 100000
000000  BIT08= 400000
000000  BIT07= 200000
000000  BIT06= 100000
000000  BIT05= 400000
000000  BIT04= 200000
000000  BIT03= 100000
000000  BIT02= 400000
000000  BIT01= 200000
000000  BIT00= 100000

.equ BIT09.BIT9
.equ BIT08.BIT8
.equ BIT07.BIT7
.equ BIT06.BIT6
.equ BIT05.BIT5
.equ BIT04.BIT4
.equ BIT03.BIT3
.equ BIT02.BIT2
.equ BIT01.BIT1
.equ BIT00.BIT0

: *BASIC CPU TRAP VECTOR ADDRESSES
000004  ERRVEC= 4  : TIME OUT AND OTHER ERRORS
000013  RESVEC= 10 : RESERVED AND ILLEGAL INSTRUCTIONS
000014  BTVEC= 14  : "T" BIT
000014  JTVEC= 14  : TRACE TRAP
000014  BTVEC= 20  : BREAKPOINT TRAP (BPT)
000020  IOTVEC= 20  : INPUT/OUTPUT TRAP (IOT) **SCOPE**
000024  PWAVEC= 24  : POWER FAIL
000030  EMTRVEC= 30  : EMULATOR TRAP (EMT) **ERROR**
000034  TRAPVEC= 34  : "TRAP" TRAP
000036  TKVEC= 60  : TTY KEYBOARD VECTOR
000036  TPVEC= 64  : TTY PRINTER VECTOR
000040  PIROVEC=240 : PROGRAM INTERRUPT REQUEST VECTOR
000060  SBTTL TRAP CATCHER

000090  =0

: *ALL UNUSED LOCATIONS FROM 4 - 776 CONTAIN A ",+2,HALT"
: *SEQUENCE TO CATCH ILLEGAL TRAPS AND INTERRUPTS
: *LOCATION 0 CONTAINS 0 TO CATCH IMPROPERLY LOADED VECTORS

000074  BSB 174
000074  DISREG: .WORO 0  : SOFTWARE DISPLAY REGISTER
000074  SWREG: .WORO 0  : SOFTWARE SWITCH REGISTER
000076  .SBTTL STARTING ADDRESS(ES)
000137  001542  JMP *START  ; JUMP TO STARTING ADDRESS OF PROGRAM
000137  .SBTTL ACT11 HOOKS

: *********************************************
: ACKS REQUIRED BY ACT11
: $PC=.
: SAVE PC
```
:1) SET LOC. 46 TO ADDRESS OF $ENDAD IN $EO2

:2) SET LOC. 52 TO ZEPC

:: RESTORE PC
### COMMON TAGS

<table>
<thead>
<tr>
<th>Tag</th>
<th>Type</th>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$CMTAG</td>
<td>.WORD</td>
<td>DC1100</td>
<td>START OF COMMON TAGS</td>
</tr>
<tr>
<td>$PASS</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS PASS COUNT</td>
</tr>
<tr>
<td>STEST</td>
<td>.BYTE</td>
<td>0</td>
<td>CONTAINS THE TEST NUMBER</td>
</tr>
<tr>
<td>$ERFLG</td>
<td>.BYTE</td>
<td>0</td>
<td>CONTAINS ERROR FLAG</td>
</tr>
<tr>
<td>$LST</td>
<td>.BYTE</td>
<td>0</td>
<td>CONTAINS SUBTEST ITERATION COUNT</td>
</tr>
<tr>
<td>$LADDR</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS SCOPE LOOP ADDRESS</td>
</tr>
<tr>
<td>$LRERR</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS SCOPE RETURN FOR ERRORS</td>
</tr>
<tr>
<td>$LATT</td>
<td>.BYTE</td>
<td>0</td>
<td>CONTAINS TOTAL ERRORS DETECTED</td>
</tr>
<tr>
<td>$ITLB</td>
<td>.BYTE</td>
<td>0</td>
<td>CONTAINS ITEM CONTROL BYTE</td>
</tr>
<tr>
<td>$ITEMA</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS MAX ERRORS PER TEST</td>
</tr>
<tr>
<td>$ERRIN</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS PC OF LAST ERROR INSTRUCTION</td>
</tr>
<tr>
<td>$GADDR</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ADDRESS OF 'GOOD' DATA</td>
</tr>
<tr>
<td>$BADR</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ADDRESS OF 'BAD' DATA</td>
</tr>
<tr>
<td>$GDATA</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS GOOD DATA</td>
</tr>
<tr>
<td>$BADATA</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS BAD DATA</td>
</tr>
<tr>
<td>$RST</td>
<td>.WORD</td>
<td>0</td>
<td>RESERVED--NOT TO BE USED</td>
</tr>
<tr>
<td>$AUTOB</td>
<td>.BYTE</td>
<td>0</td>
<td>AUTOMATIC MODE INDICATOR</td>
</tr>
<tr>
<td>$INTAG</td>
<td>.BYTE</td>
<td>0</td>
<td>INTERRUPT MODE INDICATOR</td>
</tr>
<tr>
<td>$DISP</td>
<td>.WORD</td>
<td>0</td>
<td>ADDRESS OF SWITCH REGISTER</td>
</tr>
<tr>
<td>$DISPID</td>
<td>.WORD</td>
<td>0</td>
<td>ADDRESS OF DISPLAY REGISTER</td>
</tr>
<tr>
<td>$SHK</td>
<td>.WORD</td>
<td>0</td>
<td>ITY KBD STATUS</td>
</tr>
<tr>
<td>$STB</td>
<td>.WORD</td>
<td>0</td>
<td>ITY KBD BUFFER</td>
</tr>
<tr>
<td>$TTY</td>
<td>.WORD</td>
<td>0</td>
<td>ITY PRINTER STATUS REG. ADDRESS</td>
</tr>
<tr>
<td>$TBY</td>
<td>.WORD</td>
<td>0</td>
<td>ITY PRINTER BUFFER REG. ADDRESS</td>
</tr>
<tr>
<td>$NB</td>
<td>.BYTE</td>
<td>0</td>
<td>CONTAINS NULL CHARACTER FOR FILLS</td>
</tr>
<tr>
<td>$FILLS</td>
<td>.BYTE</td>
<td>0</td>
<td>CONTAINS # OF FILLER CHARACTERS REQUIRED</td>
</tr>
<tr>
<td>$FILL</td>
<td>.BYTE</td>
<td>0</td>
<td>INSERT FILL CHAR AFTER A &quot;LINE FEED&quot;</td>
</tr>
<tr>
<td>$AR2</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ADDRESS FROM WHICH ($REGD) WAS OBTAINED</td>
</tr>
<tr>
<td>$REGC</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($REGC)+0</td>
</tr>
<tr>
<td>$REGD</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($REGD)+1</td>
</tr>
<tr>
<td>$REGE</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($REGE)+2</td>
</tr>
<tr>
<td>$REGF</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($REGF)+3</td>
</tr>
<tr>
<td>$REGG</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($REGG)+4</td>
</tr>
<tr>
<td>$REGH</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($REGH)+5</td>
</tr>
<tr>
<td>$REGI</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($REGI)+6</td>
</tr>
<tr>
<td>$REGJ</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($REGJ)+7</td>
</tr>
<tr>
<td>$REGK</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($REGK)+8</td>
</tr>
<tr>
<td>$REGL</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($REGL)+9</td>
</tr>
<tr>
<td>$REGM</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($REGM)+10</td>
</tr>
<tr>
<td>$RENG</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($RENG)+11</td>
</tr>
<tr>
<td>$REGH</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($REGH)+12</td>
</tr>
<tr>
<td>$RENI</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($RENI)+13</td>
</tr>
<tr>
<td>$REGJ</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($REGJ)+14</td>
</tr>
<tr>
<td>$REGK</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($REGK)+15</td>
</tr>
<tr>
<td>$RENG</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($RENG)+16</td>
</tr>
<tr>
<td>$REGH</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($REGH)+17</td>
</tr>
<tr>
<td>$RENI</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($RENI)+18</td>
</tr>
<tr>
<td>$REGJ</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($REGJ)+19</td>
</tr>
<tr>
<td>$REGK</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($REGK)+20</td>
</tr>
<tr>
<td>$RENG</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($RENG)+21</td>
</tr>
<tr>
<td>$REGH</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($REGH)+22</td>
</tr>
<tr>
<td>$RENI</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($RENI)+23</td>
</tr>
<tr>
<td>$REGJ</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($REGJ)+24</td>
</tr>
<tr>
<td>$REGK</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($REGK)+25</td>
</tr>
<tr>
<td>$RENG</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($RENG)+26</td>
</tr>
<tr>
<td>$REGH</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($REGH)+27</td>
</tr>
<tr>
<td>$RENI</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($RENI)+28</td>
</tr>
<tr>
<td>$REGJ</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($REGJ)+29</td>
</tr>
<tr>
<td>$REGK</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($REGK)+30</td>
</tr>
<tr>
<td>$RENG</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($RENG)+31</td>
</tr>
<tr>
<td>$REGH</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($REGH)+32</td>
</tr>
<tr>
<td>$RENI</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($RENI)+33</td>
</tr>
<tr>
<td>$REGJ</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($REGJ)+34</td>
</tr>
<tr>
<td>$REGK</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($REGK)+35</td>
</tr>
<tr>
<td>$RENG</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($RENG)+36</td>
</tr>
<tr>
<td>$REGH</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($REGH)+37</td>
</tr>
<tr>
<td>$RENI</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($RENI)+38</td>
</tr>
<tr>
<td>$REGJ</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($REGJ)+39</td>
</tr>
<tr>
<td>$REGK</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($REGK)+40</td>
</tr>
<tr>
<td>$RENG</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($RENG)+41</td>
</tr>
<tr>
<td>$REGH</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($REGH)+42</td>
</tr>
<tr>
<td>$RENI</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($RENI)+43</td>
</tr>
<tr>
<td>$REGJ</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($REGJ)+44</td>
</tr>
<tr>
<td>$REGK</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($REGK)+45</td>
</tr>
<tr>
<td>$RENG</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($RENG)+46</td>
</tr>
<tr>
<td>$REGH</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($REGH)+47</td>
</tr>
<tr>
<td>$RENI</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($RENI)+48</td>
</tr>
<tr>
<td>$REGJ</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($REGJ)+49</td>
</tr>
<tr>
<td>$REGK</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($REGK)+50</td>
</tr>
<tr>
<td>$RENG</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($RENG)+51</td>
</tr>
<tr>
<td>$REGH</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($REGH)+52</td>
</tr>
<tr>
<td>$RENI</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($RENI)+53</td>
</tr>
<tr>
<td>$REGJ</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($REGJ)+54</td>
</tr>
<tr>
<td>$REGK</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($REGK)+55</td>
</tr>
<tr>
<td>$RENG</td>
<td>.WORD</td>
<td>0</td>
<td>CONTAINS ($RENG)+56</td>
</tr>
</tbody>
</table>

---

**This table contains various common storage locations used in the program.**
EVEN

;RK11 REGISTERS
;IF FOR ANY REASON THE REGISTER ADDRESSES ARE DIFFERENT FROM THESE
;GIVEN BELOW, THE CONTENTS OF THE APPROPRIATE POINTERS SHOULD BE
;MODIFIED SO THAT THE CORRECT ADDRESS IS USED.

EVEN
RKDS: 177400
RKER: 177402
RKCS: 177404
RKC: 177406
RKGA: 177410
RKDA: 177412
RKDB: 177416

TAGS AND GENERAL DATA AREA

FTITLE: 0
;FLAG FOR PRINTING PROGRAM TITLE
TIMER: 0
;TIMER REGISTER
RKPRI: 200
;CONTAINS THE CPU LEVEL AT WHICH
;RK11 NORMALLY INTERRUPTS. THIS WORD
;SHOULD BE CHANGED IF RK11 IS DESIGNATED
;A BR LEVEL OTHER THAN 6. E.G. IF IT IS CHANGED
;TO 6, THIS WORD SHOULD BE CHANGED TO 24C.
RKVEC: 220
;CONTAINS THE NORMAL VECTOR ADDRESS TO WHICH
;RK11 INTERRUPTS. IF THIS IS NOT SO, CHANGE
;THIS WORD TO CONTAIN MODIFIED VECTOR ADDRESS.
SBTTL  ERROR POINTER TABLE

:THIS TABLE CONTAINS THE INFORMATION FOR EACH ERROR THAT CAN OCCUR.
:THE INFORMATION IS OBTAINED BY USING THE INDEX NUMBER FOUND
:LOCATION ITEMS. THIS NUMBER INDICATES WHICH ITEM IN THE TABLE IS PERTINENT.
:*NOTE:* IF ITEM IS 0 THE ONLY PERTINENT DATA IS "ERR".
:*NOTE:* EACH ITEM IN THE TABLE CONTAINS 4 POINTERS EXPLAINED AS FOLLOWS:

*  EM  :POINTS TO THE ERROR MESSAGE
*  DM  :POINTS TO THE DATA HEADER
*  DT  :POINTS TO THE DATA TABLE
*  DF  :POINTS TO THE DATA FORMA

ERRATB:

THE ERROR ITEMS TABLE CONSISTS OF ALL THE POSSIBLE ERROR MESSAGES.
USED IN THIS PROGRAM AN ERROR CALL IN THE PROGRAM CORRESPONDS TO
THE ITEM NUMBER IN THE ERROR TABLE. THUS 'ERROR 1' IN THE
PROGRAM CORRESPONDS TO 'ITEM 1' IN THE ERROR TABLE.
'EM***' IS THE POINTER TO THE ERROR MESSAGE WHICH WILL BE TYPED
OUT IN CASE THAT ERROR WERE TO OCCUR. THUS FOR 'ERROR 1' THE ERROR
MESSAGE TYPE OUT WILL BE 'TYPE OUT ON RK11 REG'.
'DM***' IS THE POINTER TO THE HEADER BLOCK WHICH WILL BE TYPED OUT
IMMEDIATELY AFTER THE ERROR MESSAGE.
'DT***' SERVES AS A POINTER TO THE MEMORY LOCATIONS WHERE
THE INFORMATION RELEVANT TO THE ERROR TYPE 1'S (LIKE PC, CONTENTS
OF REGS ETC.) WILL BE PICKED UP FROM.
THE LAST ROW CONTAINING 'D' SERVES AS A TERMINATOR.
EXAMPLE:
IF ON RUNNING THIS PROGRAM A TIMEOUT WERE TO OCCUR ON ADDRESSING REG
177400, BECAUSE OF SOME FAULT*, THE FOLLOWING TYPE OUT WOULD
OCCUR ON THE TELETYPE.

TIME OUT ON RK11 REG
PC  REG
*****  177400

*NOTE THAT ***** WOULD BE THE ACTUAL PC WHERE 'ERROR 1' IS LOCATED.

THE ERROR HANDLER IS LOCATED AT 'ERROR'. THE ERROR CALL IS AN 'EM***'
INSTRUCTION WITH ITS LOWER BYTE ENCODED TO PROVIDE INDEXING TO THE
ITEMS IN THE ERROR TABLE.

*NOTE: 'ERROR 1' IS 104001
*ERROR 126' IS 104126 ETC.

ERRATB ITEMS TABLE
ERROR POINTER TABLE

:ITEM 1
EM1 : TIME OUT ON R2!1 REG
DN1 : PC REG
DT1 : SERRPC $REGO

:ITEM 2
EM2 : REGISTER NOT CLEARED
DN2 : PC REGADD REGAD
DT2 : SERRPC $REGO $REGI

:ITEM 3
EM3 : RKCS ERROR
DN3 : PC WROTE READ
DT2 : SERRPC $REGO $REGI

:ITEM 4
EM4 : RKCS ERROR-ON WRITING READ ONLY BITS
DN4 : PC EXPECT REGAD
DT2 : SERRPC $REGO $REGI

:ITEM 5
EM5 : BUS INIT DID NOT CLEAR RKCS
DN5 : PC REGV
DT1 : SERRPC $REGO

:ITEM 6
EM6 : 'CNTAL RESET' DIDN'T CLEAR RKCS, ON SETING GO
DN6 : PC REGV
DT1 : SERRPC $REGO

:ITEM 7
EM7 : 'CNTAL RDY' DIDN'T SET AFTERT CONTROL RESET
DN7 : PC RKCS RKR RKDS
DT26 : SERRPC $REGO $REGI $REG2

:ITEM 10
EM10 : REGISTER NOT CLEARED
DN10 : PC REGADD REGAD
DT10 : SERRPC $REGO $REGI
<table>
<thead>
<tr>
<th>ITEM</th>
<th>CODE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>00142D 010201</td>
<td>BKWE ERROR</td>
</tr>
<tr>
<td></td>
<td>0132 011205</td>
<td>WRITE READ</td>
</tr>
<tr>
<td></td>
<td>011504 001440</td>
<td>ERRPC REG1</td>
</tr>
<tr>
<td>12</td>
<td>00140D 011101</td>
<td>UNEXPECTED PK11 INTERRUPT</td>
</tr>
<tr>
<td></td>
<td>014D 011530</td>
<td>PC</td>
</tr>
<tr>
<td></td>
<td>0121 001440</td>
<td>ERRPC</td>
</tr>
<tr>
<td>13</td>
<td>00141D 011706</td>
<td>BKBA ERROR</td>
</tr>
<tr>
<td></td>
<td>011504 001440</td>
<td>PC WRITE READ</td>
</tr>
<tr>
<td></td>
<td>011530 001440</td>
<td>ERRPC REG1</td>
</tr>
<tr>
<td>14</td>
<td>00142D 010721</td>
<td>CNTAL RESET DID NOT CLEAR REGISTER</td>
</tr>
<tr>
<td></td>
<td>011530 001440</td>
<td>PC REGADD RECVD</td>
</tr>
<tr>
<td></td>
<td>0121 001440</td>
<td>ERRPC REG1 REG2</td>
</tr>
<tr>
<td>15</td>
<td>001432 010773</td>
<td>BKBA ERROR</td>
</tr>
<tr>
<td></td>
<td>0134 011706</td>
<td>PC WRITE READ</td>
</tr>
<tr>
<td></td>
<td>011504 001440</td>
<td>ERRPC REG1</td>
</tr>
<tr>
<td>16</td>
<td>001432 011321</td>
<td>BKCS ALTERED ON CLEARING 'REG-BYTE'</td>
</tr>
<tr>
<td></td>
<td>011210 001440</td>
<td>PC REGADD RECVD</td>
</tr>
<tr>
<td></td>
<td>011534 001440</td>
<td>ERRPC REG1 REG2</td>
</tr>
<tr>
<td>17</td>
<td>00142D 011206</td>
<td>BUS INIT DIDN'T CLEAR REGISTER</td>
</tr>
<tr>
<td></td>
<td>014C 011504</td>
<td>PC REGADD RECVD</td>
</tr>
<tr>
<td></td>
<td>001460 000000</td>
<td>ERRPC REG1</td>
</tr>
<tr>
<td>18</td>
<td>001462 011042</td>
<td>ADDRESSING ERROR - TRIED TO ADDRESS REG1, GOT REG2</td>
</tr>
<tr>
<td></td>
<td>001462 011740</td>
<td>PC REG1  REG1 REG2 (REG1) (REG2)</td>
</tr>
<tr>
<td></td>
<td>001460 000000</td>
<td>ERRPC REG1 REG2 REG3</td>
</tr>
</tbody>
</table>
### ERROR POINTER TABLE

<table>
<thead>
<tr>
<th>ITEM</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>TRIED TO CLEAR 'REG-BYTE', CHANGED 'REG-BYTE2'</td>
</tr>
<tr>
<td>22</td>
<td>DID NOT CLEAR RKCS LO BYTE</td>
</tr>
<tr>
<td>23</td>
<td>DID NOT CLEAR RKCS HI BYTE</td>
</tr>
<tr>
<td>24</td>
<td>TRIED TO CLEAR RKCS 'BYTE', CHANGED 'REGIS'</td>
</tr>
<tr>
<td>25</td>
<td>FAILED TO CLEAR 'REG-BYTE'</td>
</tr>
</tbody>
</table>

### INITIALIZE COMMON TAGS

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLEAR THE COMMON TAGS (SCMTAG) AREA</td>
</tr>
<tr>
<td>FIRST LOCATION TO BE CLEARED</td>
</tr>
<tr>
<td>CLEAR MEMORY LOCATION</td>
</tr>
<tr>
<td>DONE?</td>
</tr>
<tr>
<td>LOOP BACK IF NO</td>
</tr>
<tr>
<td>SETUP THE STACK POINTER</td>
</tr>
<tr>
<td>INITIALIZE A FEW VECTORS</td>
</tr>
<tr>
<td>INITIALIZE NUMBER OF ITERATIONS</td>
</tr>
</tbody>
</table>
Initialize the common tags

Clear the escape on error address

Allow one error per test

Initialize the loop address for scope

Setup the error loop address

Set up a hardware switch register. If not found or it is equal to 'A', setup for a software switch register.

Save error vector

Set up error vector

Setup a hardware switch register and a hardware display register

Try to reference hardware switch

Branch if no timeout trap occurred

Branch if no timeout

Branch if no timeout

Save trap return

Point to software switch

Restore error vector

Are we in act11 automatic mode?

If yes, skip title

Type program name

The name of the program if first pass

First time?

Branch if no

Type ASCII string

Get value for software switch register

Are we running under XXDF AC?

Branch if yes

Branch if no

Software switch reg selected?

Get soft-sw settings

Set auto-mode indicator

Get over the ASCIZ

Type ASCII<br> (CRLF)/RA11 LOGIC TEST /(<15><12>)/MAINDEC-11-DZKJ-E CRLF

Set time out vector for unexpected time outs

Set up RK11 interrupt vector for unexpected interrupts from RK11

Go to test 1

This routine handles unexpected time outs

Save PC where time out occurred

Restore stack pointer

Type ASCII string

Get over the ASCII
MO2

MC: 1-0294-5, RK1: BASIC LOGIC TEST 1 MAC:11 301146) 06-JUN-77 14:40 PAGE 25

GET VALUE FOR SOFTWARE SWITCH REGISTER

;:65$: .ASCIIZ "<15><12>/UNEXPECTED TIME OUT AT PC= /

64$: MOV RO,-(SP) ;SET UP FOR TYPING OUT PC

TYPE

10: GO TYPE OUT OCTAL PC

HALT

JMP #START

;:THIS ROUTINE HANDLES UNEXPECTED INTERRUPTS FROM RK1:

SET W AND 10 FOR LOOPING ON ERROR

; AND LOOPING ON TEST IN WHICH TIMEOUT

; OCCURRED, ARE PROVIDED.

BADINT: MOV (SP) RO ;SAVE PC WHERE INTERRUPT OCCURRED

SFR - (RO)

TST - (RO)

BIT, #0000, BSWR ;INHIBIT ERROR TYPEOUT?

BNE 1$ ;YES, DON'T TYPE OUT

TYPE $CRFL

TYPE

EM43 ;TYPE 'UNEXPECTED RK11 INTERRUPT'

; TYPE ' AT PC= '

:TYPE ASCIZ STRING

; GET OVER THE ASCIZ

TYPE ,65$

BR 64$ ;GET OVER THE ASCIZ

:65$: .ASCIIZ " AT PC=/

64$: MOV RO,-(SP) ;SET UP FOR TYPING OUT PC

TYPE

10: GO TYPE OCTAL PC WHERE BAD

INTERUPT OCCURED

LOOP ON ERROR?

BEQ 2$ ;NO, BRANCH

CPP (SP)+,(SP)+

JMP $LPADR

;GO TO THE STARTING ADDRESS OF

;THE TEST THAT GAVE UNEXPECTED INTERRUPT.

;LOOP ON TEST?

BEQ 3$ ;NO, BRANCH

RTI

;YES, LOOP. GO BACK WHEN U INTERRUPTED FROM:

;EXPECTED INTERRUPT OCCURED AS

;INDICATED IN THE TYPE OUT.

;U CAN LOOP ON ERROR, TEST, OR INHIBIT TYPEOUT BY

;SETTING APPROPRIATE SWITCHES.

;GO BACK TO THE START OF THE

;PROGRAM. THUS PRESSING CONTINUE

;AFTER THE ABOVE HALT WILL

;RESTART THE PROGRAM

:RESTART AFTER POWER FAIL

;THE PROGRAM WOULD RESTART HERE IF POWER CAME BACK AFTER A FAILURE.

PFSTRT:

:TYPE ,65$

;:TYPE ASCIZ STRING

BR 64$ ;GET OVER THE ASCIZ

:65$: .ASCIIZ "<15><12>/PWR UP,RESTART/

64$: CLR RO
;************************************************************
;** TEST 1 **
;** TEST THAT ALL RK11 REGISTERS CAN BE REFERENCED **
;** THIS TEST CHECKS IF EVERY RK11 REGISTER CAN BE **
;** REFERENCED WITHOUT TIMING OUT. IF A TIME OUT OCCURS THE ERROR IS **
;** REPORTED & AN ERROR FLAG (R2) IS INCREMENTED. IF THERE WAS **
;** AN ERROR DURING THIS TEST, THE ENTIRE PROGRAM IS ABORTED **
;************************************************************

;ST1: SCOPE
CLR R2
MOV #1, $LPERR  ; SET RETURN ADDRES FOR LUP
MOV #ERR (SW9)  ; ON ERROR (SW9)
MOV #TIMOUT, #4  ; SET UP ADDRESS FOR TIMEOUT VECTOR
MOV #7, R0      ; INITIALIZE RD TO KEEP TRACK OF REGIS REFERENCED
MOV #RDS,R1     ; INITIALIZE RI WITH RKDS ADDRESS
T1: TST @((R1)+1)  ; REFERENCED THE REGISTER;
     IF IT CAN'T BE TIMOUT TRAP WILL OCCUR
     SHIFT THE POINTER
     ALL REGISTERS REFERENCED?
     IF NOT, LOOP BACK & REFERENCE NEXT REGISTER
     HAS THERE AN ERROR?
     NO BRANCH
     TYPE 55$  ; TYPE ASCIZ STRING
     GET OVER THE ASCIZ
     ASCIZ (15<12>/PROG ABORTED/
84$: JMP $GET42  ; IF YES, ABORT THIS ENTIRE TEST

;ST2: SCOPE
CLR R1
INC R0
BNE T1          ; EXIT
T2: TST R2       ; GET ADDRESS OF REGISTER THAT TIMED OUT
     ERROR 1      ; TIMED OUT WHEN REFERENCING RK11
     REGISTERS
     REPOSITION POINTER TO THE NEXT REGISTER ADDRESS
     SET FLAG INDICATING ERROR
     BR T1+2       ; BRANCH BACK & REFERENCE THE NEXT REGISTER

;************************************************************
;** TEST 2 **
;** CHECK RK11 INITIALIZATION **
;** THIS TEST CHECKS THAT THE CONTROLLER LOGIC IS INITIALIZED **
;** CORRECTLY. RKUC, RKDA, RKDB, RKOB SHOULD BE CLEAR AND **
;** RKCS SHOULD HAVE 'CNRAL RDY' BIT SET. **
;************************************************************

;ST2: SCOPE
RESET
MOV #6, R0       ; ISSUE A BUS INIT
MOV #COUNT FOR R REGISTERS
MOV #RAK,R1      ; INITIALIZE ADDRES
I$: CMP RKCS,R1  ; IS IT RKCS?
**TEST 6**: CHECK RKCS SSE EXB, SMB, IBA BITS - 9, 9, 10, 11

- This test checks if the SSE, SMB, and IBA bits can be written and read back correctly.

```assembly
; INITIALIZE BIT TO BE WRITTEN IN RKCS
MOV #40H, R1
MOV RKCS, R2
MOV R1, R2
GET RKCS

; WRITE THAT BIT IN RKCS
BIT #20H, R0
MASK CTRL ADY BIT
CMP R1, R0
HAS THE READ BIT SAME AS THE WRITTEN BIT?
BEQ 2S
YES BRANCH, OTHERWISE REPORT ERROR
MOV R1, $RREG
GET EXPECTED RKCS
MOV #R2, $RREG
GET RECEIVED RKCS
ERROR 3
BIT THAT WAS WRITTEN AS IN $RREG

; SHIFT TO WRITE NEXT BIT
ASL R1
HAVE YOU CHECKED ALL BITS 9, 9, 10, 11, ...?
BNE 1S
IF NOT, LOOP BACK & CHECK THE NV+ BT.
```

**TEST 7**: CHECK READ ONLY BITS OF RKCS

- This test checks that trying to set the unused bit or the read only bits does not set them or affect any other bits in RKCS.

```assembly
; TRY SETTING THE UNUSED BIT & RC
MOV RKCS, R0
MOV #100000H, R0

; ONLY BITS
MOV R2, R1
GET RKCS

; MASK BIT 12
BIT #10000H, R1
IS 'RDY' BIT? NO OTHER BIT SHOULD BE SET.
CMP #200H, R1
LOW EXIT

; GET EXPECTED RKCS
MOV #R2, $RREG
GET RECEIVED RKCS
MOV #R2, $RREG
GET RECEIVED RKCS
ERROR 4
TRIED TO SET UNUSED & RO ONLY BITS OF RKCS
SHOULD NOT HAVE AFFECTED ANY BITS
```

**TEST 8**: CHECK THAT 'GO' BIT (0) CAN BE SET

- This test checks that the 'GO' bit can be set, by performing control reset and seeing that the EXB & IBA set previously were cleared.

```assembly
; CONTROL RESET & SEEING THAT THE EXB & IBA SET PREVIOUSLY WERE CLEARED.

; INITIALIZE BT TO BE WRITTEN IN RKCS
MOV #40H, R1
MOV RKCS, R2
MOV R1, R2
GET RKCS

; WRITE THAT BIT IN RKCS
BIT #20H, R0
MASK CTRL ADY BIT
CMP R1, R0
HAS THE READ BIT SAME AS THE WRITTEN BIT?
BEQ 2S
YES BRANCH, OTHERWISE REPORT ERROR
MOV R1, $RREG
GET EXPECTED RKCS
MOV #R2, $RREG
GET RECEIVED RKCS
ERROR 3
BIT THAT WAS WRITTEN AS IN $RREG

; SHIFT TO WRITE NEXT BIT
ASL R1
HAVE YOU CHECKED ALL BITS 9, 9, 10, 11, ...?
BNE 1S
IF NOT, LOOP BACK & CHECK THE NV+ BT.
```
64%: MOV RKC5, R1 ; SET ALL BITS EXCEPT GO
MOV #756, JR1
CLR RO ; ISSUE BUS INIT
CLR RO ; CHECK IF RK5 WAS CLEARED?
CMP #200, JR1
BEO 16 ; BRANCH OTHERWISE REPORT ERR
MOV JR1, $REPG ; GET RK5
MOV SR1, $REGO ; BUS INIT DID NOT CLEAR RK5
SET IBA & EXB IN RK5
MOV #5000, JR1 ; SET GO, CONTROL RESET
INC RO ; KEEP TIME
INC RO ; HAVE YOU WAITED LONG FOR CTRL RDY?
10: TO SET?
BMI 13 ; IF YES, BRANCH & REPORT ERR
TSTB RO ; WAS CTRL RDY SET?
BFL 28 ; IF NOT LOOP BACK & WAIT FOR IT
CMP #200, JR1 ; IF CTRL RDY WAS SET, CHK IF 'CTRL
BEQ TST11 ; ' CLEARED IBA & EXB BITS
ERROR 6 ; IF YES, EXIT. OTHERWISE ERROR
MOV JR1, $REPG ; GET RK5
GO BIT COULD NOT BE SET OR FAULT IN
TST11 ; 'INIT' GENERATING LOGIC
 ; EXIT
BR TST11 ; GET RK5, ER, DS
3$: JSR P, GT3RG ; CONTROL READY DID NOT SET AFTER
MOV #64$, -(SP) ; CONTROL RESET
RTI

===============================================
; TEST 11 CHECK RKCS WITH A COUNT PATTERN
; THIS TEST CHECKS THAT RKCS CAN BE CLEARED FROM 7567, THEN A COUNT
; PATTERN FROM 2 TO 7777 IS RUN. NOTE: ALL PATTERNS WITH BIT 0 SET
; (GO BIT) ARE AVOIDED SO THAT RK11 MAY NOT START AN UNDESIRED OPERATION.
; JR1 CONTAINS THE COUNT PATTERN THAT WAS WRITTEN

15$: SCOPE #10, $TIMES ; DO 10 ITERATIONS
MOV #10, $TIMES
MOV #340, -(SP)
MOV #64$, -(SP)
RTI

64%: MOV RKC5, RO ; SET ALL BITS IN RKCS except GO
MOV #756, JR0
CLR RO ; CLEAR RKCS
CLR RO ; WAS IT CLEARED?
CMP #200, JR0
BEO 16 ; YES BRANCH
MOV JR0, $REPG ; GET ADDR OF RKCS
MOV $RECO, JR0 ; NO, GET RKCS
ERROR 10 ; RKCS COULD NOT BE CLEARED
WRITE THIS BIT IN RKCS
MOV #0, JR1
MOV $2, JR1
MOV #2, JR1
MOV $3, JR1
MOV #3, JR1
MOV $5, JR1
WRITE IT
MOV R1, JR1
MOV R2, JR1
GET BIT THAT WAS WRITTEN
MOV #200, R2
MOV JR0, R3
MOV JR0, R3
F03

0-11-C21J-E, AK11 BASIC LOGIC TEST 1  MACY 11 10-1446 06-JUN-77 14:40 PAGE 31
0-11-C21J-E, 19-APR-77 08:14

T:11 CHECK AKCS WITH A COUNT PATTERN

CMP R2,R3 ; WAS THAT BIT WRITTEN CORRECTLY?
BEQ 3 ; YES, BRANCH
MOV R2,$REG0 ; GET EXPCTD WORD
MOV R3,$REG1 ; GET AKCS REGD
ERROR 3 ; DID NOT READ BK 6 TO THE BIT THAT
; WAS WRITTEN
INC R5 ; EXIT
BEQ TST12 ; GENERATE N.T. PATTERN TO BE WRITTEN
CMP #1000D,R1 ; ALL PATTERNS WRITTEN?
BNE 2 ; IF NOT, LUP BK & CK NEXT PATTERN

TST12: SCOPE
MOV #1,R0 ; INITIALIZE RO FOR THE BIT TO BE WRITTEN IN RKC
MOV AKWC,R1 ; SET UP RETURN ADDRESS FOR
MOV #15,$LPERR ; LUPING ON ERROR (SW 9)
MOV R0,RO
MOV #1,R1 R2
BEQ #1,R0,2 ; WAS IT WRITTEN CORRECTLY
MOV R0,$REG0 ; YES, BRANCH, OTHERWISE, ERROR
MOV R2,$REG1 ; GET EXPCTD AKWC BIT THAT WAS WRITTEN
MOV R1,R2 ; GET AKWC (THAT WAS READ BACK)
ERROR 11 ; DID NOT READ BACK THE BIT "HAT"
; WAS WRITTEN IN AKWC
; SHIT TO WRITE THE NEXT BIT
BNE 1 ; IF ALL THE BITS HAVE NOT BEEN
; DONE, LOOP BACK

TST13: SCOPE
MOV #10,$TIMES ; DO 10 ITERATIONS
MOV AKWC,R0
MOV #17777777,R0 ; SET ALL BITS IN AKWC
CLR #RD ; CLEAR AKWC
TST #RD ; WAS IT CLEAR?
BEQ 1 ; YES, BRANCH
MOV R0,$REG0 ; GET ADDRESS OF AKWC
MOV #RD,R0 ; NO, GET AKWC
ERROR 10 ; AKWC COULD NOT BE CLEARED

1: CLR R1 ; INITIALIZE COUNT PATTERN
MOV #5,R5
MOV #15,$LPERR
MOV R1,#RD
MOV R0,R2 ; WRITE THE PATTERN IN THE REGISTER
MP #1, R2 ; WAS IT WRITTEN CORRECTLY?
03

**CHECK RKDA WITH A COUNT PATTERN**

2$: MOV R1, ARO        ;WRITE THE PATTERN IN THE REGISTER
    MOV ARO, R2
    CMP R1, A2
    BEQ 3$
    MOV R2, $REG1
    MOV R1, $REG0
    MOV R2, $REG1
    MOV R2, $REG1
    MOV R2, $REG1
    MOV R2, $REG1
    MOV R2, $REG1
    INC R5
    BNE 10729
    :EXIT
    BNE R1
    :INCREMENT COUNT PATTERN IF NOT
    :DONE WITH ALL

*TEST 20: CHECK THAT RKWC, RKBA, RKDA CAN BE CLEARED BY BUS INIT*

*THIS TEST CHECKS THAT RKWC, RKBA AND RKDA CAN BE CLEARED BY BUS INIT.*

*RESET INSTRUCTION IS USED*

10729: SCOPE
        MOV RKWC, R0        ;INITIALIZE RO TO PRINT TO RKWC
        MOV R0, R2
        MOV 3, R1
        :SET UP COUNT FOR 3 REGISTERS TO BE CARED
        MOV R1, R3
        MOV R2, $REG1
        MOV R2, $REG1
        INC R1
        BNE R1
        INC RKBA
        BNE 1$
        INC RKDA
        BNE 2$
        ISSUE A BUS INIT
        WAS THE REGISTER (PTD TO BY R2, CLEARED)
        BEQ 3$
        INC R3
        NO, GET ADDRESSES OF REGS THAT WERE CLEARED
        MOV R2, $REG1
        MOV $REG0
        SET CONTENTS OF REGS
        ERROR 1$
        AK0 REGISTER (ADDRES IN R0) COULDN'T
        NOT BE CLEARED BY BUS INIT
        INCCREMENT POINTER TO NEXT REGISTER
        HAVE U CHECKED ALL REGISTERS
        BNE 2$
        IF NOT, LOOP BACK & CHECK NEXT

*TEST 21: CHECK THAT RKCS, RKWC, RKBA, RKDA CAN BE CLEARED BY CONTROL RESET*

*RKCS IS SET TO 7560, RKWC, RKBA, RKDA ARE ALL SET
*TO 17777. CONTROL RESET IS DONE AND IT IS CHECKED*

*IF ALL THS REGISTERS ARE CLEARED*

4$: MOV RKCS, RS
    MOV #7560, R2
    MOV R2, $REG1
    SET ALL WRITEABLE BITS IN RKCS
    INC R3
    SET GO, CONTROL RESET
    CLR R5
    :DID CONTROL RDY SET
    BEQ R0
    :YES, BRANCH
    INC R5
    :WAITED LONG.
```
004306 001374 005434  BNE $3, R3  ; IF NOT LUP BAK & WAIT
004319 104007  ISR PC, GT3RG
004322 027456  ERRP  ; GET RKCS, RKWC, RKBA
004335 001405  CMPL $200, 3RD  ; RKDC CHK
004348 001405  MOV R0, $REG  ; CK ADRES OF RKC
004351 001405  MOV R0, $REG1  ; GET CONTENTS OF RKC
004364 001405  ERROR 4  ; CONTROL RESET DID NOT CLEAR RKC
004367 001405  MOV RKWC, R2  ; SET ALL BITS IN RKWC
004367 001405  MOV R2, R4  ; GO, DO CONTROL RESET
004367 001405  MOV R1, R2  ; THIS IS A CALL FOR THE 'CNTL.-
004367 001405  MOV R1, R2  ; RESET' ROUTINE, A CONTROL RESET IS
004367 001405  MOV R1, R2  ; DONE & AFTER A CERTAIN TIME IF
004367 001405  MOV R1, R2  ; 'CNTL.RDY' DOES NOT SET AN ERROR IS
004367 001405  MOV R1, R2  ; REPORTED. NOTE THAT THE PC IN ERROR
004367 001405  MOV R1, R2  ; IS THE PC WHERE CNTL.RESET IS
004367 001405  MOV R1, R2  ; LOCATED. THIS IS A VERY BASIC ERROR &
004367 001405  MOV R1, R2  ; IF IT OCCURS GO BACK TO TEST 10.
004367 001405  MOV R1, R2  ; WAS THE REGISTER CLEARED?
004367 001405  MOV R1, R2  ; YES, BRANCH
004367 001405  MOV R4, $REG  ; GET ADRES OF REGISTER IN ERROR
004367 001405  MOV R4, $REG1  ; GET CONTENTS OF THAT REGISTER
004367 001405  ERROR 4  ; CONTROL RESET DID NOT CLEAR THE
004367 001405  MOV R4, $REG  ; REGISTER WHOSE ADRES IS IN R4
004367 001405  INC R3  ; INCREMENT POINTER TO NEXT REGISTER
004367 001405  BNE $4  ; CHK & ALL REGS
004367 001405  BNE $4  ; IF NOT, LUP BAK & CHK THE NEXT REG
```

---

```
*----TE22  CHECK THAT EACH RK11 REGISTER IS UNIQUELY ADDRESSED
* THIS TEST CHECKS THAT EACH RK11 REGISTER CAN BE UNIQUELY ADDRESSED
* (1) RKCS RKWC, RKBA, RKDR ARE THE FIRST SET TO 17777 (17576 FOR RKCS)
* (2) REGISTER WHOSE ADDRESS IS IN R0 IS CLEARED
* (3) EVERY OTHER REGISTER IS CHECKED FOR ERRONEOUS CLEARING BECAUSE OF
* ADDRESSING ERROR. IF SO THE MULTIPLE ADDRESSING ERROR IS REPORTED
* (4) ADDRESS IN R0 IS CHANGED TO THE NEXT REGISTER & THE PROCESS IS
* REPEATED.

\*ST22: SCOPE
```

```
004410 000004  MOV $340, -(SP)  ; SET UP COUNT FOR THE # OF
004413 002746  MOV $64, -(SP)  ; REGISTERS TO BE UNIQUELY ADDRES
004416 002746  RII
004419 000002  64:
004422 002705  MOV $7, R5  ; : REGISTERS TO BE UNIQUELY ADDRES
004425 177771  MOV RKCS, R0  ; : INITIALIZE POINTER TO REGS TO BE SET
004428 001244  MOV $4, R1
004431 001270  MOV RKDC, R2
004434 001270  MOV $17576, (R2)  ; : SET BITS IN RK5
```
122 CHECK THAT EACH AKII REGISTER IS UNIQUELY ADDRESSED

1830 004459 002821 177777 3$: MOV A1 177777(R2)+ SET BITS IN RKAC
1831 004456 001374
1832 004456
1833 004456 009010
1834 004468 013782 001250
1835 004466 020002
1836 004466
1837 004467 001406
1838 004467
1839 004472 011203
1840 004474 042703 170200
1841 004474
1842 004500 027030 007576
1843 004500
1844 004504 001200
1845 004504 005722
1846 004510 017011 177775
1847 004514 002000
1848 004516 001404
1849 004516
1850 004520 011203
1851 004522 010004
1852 004522 005104
1853 004522 001007
1854 004522 005722
1855 004522 001367
1856 004522
1857 004536 005720
1858 004540 002005
1859 004540 001334
1860 004544 000415
1861 004546 010037 001162
1862 004546
1863 004552 010237 001164
1864 004552
1865 004556 011037 001166
1866 004556
1867 004562 010337 001170
1868 004562
1869 004566 104020
1870 004566
1871 004570 023702 001250
1872 004574 001744
1873 004574
1874 004576 000754

*:******** TEST 23 CHECK THAT HI & LO BYTES OF RKCS CAN BE ADDRESSED
*:THIS TEST CHECKS THAT THE HI & LO BYTES OF RKCS CAN BE
*:ADRESSED CORRECTLY
*:IF B'S IN ALL REGISTERS THAT CAN BE WRITTEN ARE SET, THEN EACH
T23: CHECK THAT HI & LO BYTES OF RCKS CAN BE ADDRESSED

; BYTE OF RCKS IS REFERENCED BY 'CLRB' & IT IS CHECKED THAT ONLY
; THAT BYTE & NO OTHER REGISTER BYTE GETS CLEARED

TST23: SCOPE

64$: MOV RCKS,R1 ; INITIALIZE PTR TO RCKS
      MOV #7777,(R1)+ ; SET ALL BITS IN RCKS

1$: MOV #17777,(R1)+ ; SET ALL BITS IN RCKS

INR R0 ; RCKS
      CALL #17777 ; RCKS

MOV RCKS,R1 ; INITIALIZE PTR TO RCKS LO BYTE

2$: CLRB @R1 ; CLER RCKS LO OR HI BYTE
      MOV @R1,R4
      MOV @RCKS,R3 ; GET RCKS WORD
      CMP 8-2,R2 ; IS CHECKING HI OR LO BYTE?
      BNE 3$ ; BRANCH IF HI BYTE
      BIC #17600,R3 ; MASK HI BYTE
      BEQ 4$ ; OK IF LO BYTE WAS CLEARED
      CLR $RGO ; GET EXPCTD RCKS.
      MOV R3,$REG1 ; GET RCKS RECVD, LO BYTE
      ERROR 22 ; ALL RCKI REGISTER'S WERE LOADED WITH
      ; I'S THEN TRIED TO ADRES & CLR RCKS
      ; LO BYTE, IT COULD NOT BE CLEARED.

3$: BIC #377,R3 ; MASK LO BYTE
      BEQ 4$ ; OK IF HI BYTE WAS CLEARED
      CLR $REGO ; GET EXPECTD RCKS, HI BYTE
      MOV R3,$REG1 ; GET WHAT WAS ACTUALLY RECVD

4$: MOV #4,R0 ; INITIALIZE COUNT FOR REST OF REGISTERS
      MOV RCKS,R5 ; INITIALIZE POINTER TO RCKS
      TST (R5)+ ; INCREMENT PTR TO NEXT REGIS
      INC R0
      BEQ 6$ ; IF YES, GO & ADRES TO CLEAR RCKS HI BYTE
      MOV $RS5,R3 ; IF NOT, GET CONTENTS OF THE REGIS
      BEQ 7$ ; BEING CHKD
      MOV R3 ; COMPLEMENT THE CONTENTS. SHOULD BE 0 FOR IT WAS
      COM R3 ; PREVIOUSLY SET TO ALL 1's IF IT'S
      BEQ 5$ ; BORNE, IF NOT REPORT ERROR
      MOV R1,$REGO ; GET RCKS-BYTE-ADRES WHICH WAS 'RIED TO
      MOV R5,$REG1 ; GET ADRES OF REGIS WHICH GOT ADRESSED
      MOV #17777,$REG2 ; GET EXPCTD CONTENTS OF REGISTER
      COM R3 ; THAT GO ADRESSED
      MOV $REGC,REG3 ; GET CONTENTS RECVD FROM THAT REGIS
ALL RK11 REGISTERS WERE LOADED WITH 1S. RKCS BYTE (ADRES UNDER 'BYTE' IN ER MSGE) WAS ADRESSED USING 'CLR' BUT REGISTER (ADRES UNDER 'REGS' IN ER MSGE) GOT CHANGED AS A RESULT.

POSITION PTR TO RKCS HI BYTE.

CHK IF BOTH HI & LO BYTES (RKCS) WERE CLEARED.

IF NOT BRANCH BACK.

CHECK THAT HI & LO BYTES OF RKWC, BA, DA CAN BE ADRESSED.

CORRECTLY, FIRST RKWC, RKBA, RKDA ARE SET TO 177777.

IT IS CHECKED THAT ONLY THAT BYTE AND NO OTHER BYTES GET CLEARED.

POINTER R2 IS INCORPORATED TO THE NEXT REGISTER-BYTE & THE PROCESS IS REPEATED. LO BYTE IS DONE FIRST, THEN HI BYTE IS DONE.

DT24: SCOPE

MOV $340, -(SP)

MOV $64$, -(SP)

RTI

MOV #6, R4

MOV RKWC, R2

MOV #3, R0

MOV RKCS, R1

MOV $7576, (R1)+

MOV $177777, (R1)+

INC R0

BNE 2$

RKBA

RKDA

CLRB 3R2

ADDRESS & CLEAR REGS BYTE UNDER "EST"

MOV R2, R0

BIC $1, R0

MOV $90, R1

BIT $1, R2

BNE 3$

BIC $177400, R1

BR 4$

MOV #377, R1

BEQ 5$

MOV R2, $REG3

TRIED TO B ADRESSED & CLEARED

BIT $1, R2

BEQ 1$

SWAP R1

MOV $10137, R1

ERROR 2$

TRIED TO ADRES & CLR A REGISTER BYTE
END OF PASS ROUTINE

:INCREMENT THE PASS NUMBER ($PASS)
:INDICATE END-OF-PROGRAM AFTER 1 PASSES THRU THE PROGRAM
:TYPE "END PASS $XXXXX" (WHERE XXXXX IS A DECIMAL NUMBER)
:IF THERE'S A MONITOR GO TO IT
:IF THERE ISN'T JUMP TO STARTI

$SOP:

$EOP:

$EOPCT:

$SENDCT:

$SENDAD:

$DOAGM:

.GT3RG: ROUTINE FOR GETTING PKCS, RKER, PKS

.GT3RG:

:SUBROUTINE FOR TRANSFERRING THE CONTENTS OF PKCS, RKER, PKS
:TO $REG0, $REG1, $REG2 RESPECTIVELY BEFORE TYPING OUT AN
:ERROR MESSAGE.

:CALL: JSR PC,GT3RG

:GET PKCS

:GET RKER

:GET PKS

:EXIT FROM "THIS SUBROUTINE"
.SBTL  GIVRG:  ROUTINE FOR GETTING RKC, RKER, RKC, RKG

; SUBROUTINE FOR TRANSFERRING CONTENTS OF RKC, RKER, RKC, RKG
; TYPE OUT AN ERROR MESSAGE.
; CALL:  JSR  PC, GIVRG

005462  E2 7F 72  30 5434
005465  21 77  37  75 66  00 17 70
005472  30 02 00 00

.SBTL  DELAY:  TIME DELAY ROUTINE

; THIS ROUTINE PROVIDES A VARIABLE TIME DELAY.  THE CALL FOR THIS
; ROUTINE IS AN ENCODED 'TRAP' INSTRUCTION.
; CALL:  DELAY N, M IS ANY OCTAL NO. FROM 1 TO 17777
; THE DELAY PROVIDED IS 7.5N US (CONVERT N TO DECIMAL) FOR 11/20
; 1.5N US FOR 11/45
; IF THE USER WANTS TO CHANGE THE DELAY TIME (EXMP: SHORTER DELAY TO
; GET A TIGHTER SCOPE LOOP) THE VARIABLE 'N' FOLLOWING 'DELAY' SHOULD
; BE CHANGED TO SUIT THE INDIVIDUAL NEED.

005474  17 76 73  30 00 00  30 12 64
005502  64 72 16  30 00 00  02 12 64
005506  00 53 37  30 12 64
005512  00 13 75
005514  30 00 02

.SBTL  CON.RESET:  CONTROL RESET ROUTINE

; CON.RESET
; THIS ROUTINE ISSUES A CONTROL RESET AND WAITS FOR
; THE 'CNTL RDY' FLAG TO SET.  WHEN THE FLAG SETS
; AN EXIT IS MADE OUT OF THE ROUTINE. IF 'CNTL-ARDY'
; DOES NOT SET WITHIN A CERTAIN TIME AN ERROR MESSAGE
; IS GIVEN.  NOTE THAT XXXXXXX IS THE PC WHERE 'CNT.RESET' OR 'CNT.RDY'
; IS CALLED.
; CALL:  CNT.RESET

.SBTL  CNT.RDY:  WAIT FOR CONTROL READY ROUTINE
::CN.RDY
::THIS ROUTINE WAITS FOR THE CONTROL READY BIT TO GET SET AND WHEN IT
::SETS EXITS OUT, IF WITHIN A CERTAIN TIME CNTRL.RDY DOES
::NOT GET SET, AN ERROR IS REPORTED. WAITING TIME IS 883 MS FOR 11:45
::175 MS FOR 11:45 WITH BIPOLAR MEMORY.
::CALL: CNT.RDY
::CN.RDY:
::MOV 1,ARCS
::ISSUE A CONTROL RESET
::MOV 8-HOOG $REG3
::SET UP COUNT
::BR 0
::SKIP OVER CN.RDY
::CN.RDY:
::CLR $REG3
::DID CNTRL.RDY SET?
::BSI 3
::YES, EXIT
::INC $REG3
::WAITED LONG?
::BNE 1
::IF NOT, GO BACK & WAIT
::BIT 85W13,SWR
::INHIBIT TYPEOUT?
::BNE 3
::IF YES, SKIP TYPECLT
::TYPE
::MSG3
::TYPE 65
::TYPE ASCIZ STRING
::BR 64
::GET OVER THE ASCIZ
::64:
::ASCIZ <15>:"12":PC=
::64:
::MOV (SP) = (SP)
::SUB 82.(SP)
::GO TYPE PC IN THE MAIN PROGR.
::TYPE
::TYPE 67
::TYPE ASCIZ STRING
::BR 66
::GET OVER THE ASCIZ
::66:
::ASCIZ / ARKC=:
::66:
::MOV ARKC= (SP)
::GET ARKC
::TYPE
::GO TYPE IT
::3:
::RTI
::RETURN FROM THIS
::ROUTINE TO THE MAIN
::PROGRAM

::THIS PART OF THE PROGRAM CONTAINS THE COMMON ROUTINES CALLED
::FROM THE SYSMAC.SML PACKAGE

::SBTL SCOPE HANDLER ROUTINE

::THIS ROUTINE CONTROLS THE LOOPING OF SUBTESTS. IT WILL INCREMENT
::AND LOAD THE TEST NUMBER($STNM) INTO THE DISPLAY REG.(DISPLAY<7:0)
::AND LOAD THE ERROR FLAG ($ERFLG) INTO DISPLAY<15:0B>
::THE SWITCH OPTIONS PROVIDED BY THIS ROUTINE ARE:
::SWL=1 LOOP TEST
::SWL=1 INHIBIT ITERATIONS
::SWL=1 LOOP ON ERROR
::SWL=1 LOOP ON TEST IN SWR<7:0>
::CALL
::SCOPE ::SCOPE=ICT
ERROR MESSAGE TYPEOUT ROUTINE

ASL RO
ADD #ERRTB, RO
MOV @RO+, RO
BEQ 3S

DEF TYPE

2S: "WORD 0
TF "SRLF
"CARRIAGE RETURN" & "LINE FEED"
MOV @RO+, RO
BEQ 5S

DEF TYPE

4S: "WORD 0
TF "SRLF
"CARRIAGE RETURN" & "LINE FEED"
MOV @RO+, RO
BNE 7S

DEF TYPE

5S: "WORD 0
TF "SRLF
"CARRIAGE RETURN" & "LINE FEED"
MOV @RO+, RO
BNE 7S

DEF TYPE

7S: MOV @RO+, @SP
TF "TYPE01"
BNE 6S
TYPE 8S
BR 2S

9S: "ASCII .
EVEN

.SBTL TTY INPUT ROUTINE

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

.SEND LSB

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

SOFTWARE SWITCH REGISTER CHANGE ROUTINE

#ROUTINE IS ENTERED FROM THE TRAP HANDLER, AND WILL
#SERVICE THE TEST FOR CHANGE IN SOFTWARE SWITCH REGISTER TRAP CALL
#WHEN OPERATING IN TTY FLAG MODE.

CPSW: CMP #SWREG, SW
IS THE SOFT-SW SELECTED?
BNE 1S

TSTB @STKS
IF NO, "DON'T WAIT AR0LNC
BPL 2S

MOV B @STKB, -(SP):
SAVE THE CHAR

BIC #11111111, SP:
"STRIP OFF THE ASCII

CMP #00000001, SP:
"IS IT A CONTROL C?

BNE 3S
NO RETURN TO USER

CMPB #AUTO, @
ARE WE RUNNING IN ALT-MODE?

BGT 4S
BRANCH IF YES

5S: "SWAP TYPE
.TYPETAB.BG
ECHO THE CONTROL-G, G

6S: "SWAP TYPE
.TYPETAB.
SAVE SWAP FOR TYPE01

7S: "SWAP TYPE
.TYPETAB.
GO TYPE-OCTAL ASCII ALL DIGITS

8S: "SWAP TYPE
.TYPETAB.
PROMPT FOR NEW SWAP

9S: "SWAP TYPE
.TYPETAB.
CLEAR COUNTER

10S: "SWAP TYPE
.TYPETAB.
THE NEW SWAP

11S: "SWAP TYPE
.TYPETAB.
"CHAR HERE"
BPL "$" : IF NOT TRY AGAIN

MOVB $51B, SF : PICK UP CHAP

BIC $101, SF : MAKE IT -BI- AS:1:

CMP $SP, $25 : IS IT A CONTROL-U
BNE 10% : BRANCH IF NOT

TYPE $CNTL : YES, ECHO CONTROL-U...

ADD $6, SP : IGNORE PREVIOUS INPUT...
BR 19% : LET'S TRY IT AGAIN

CMP $SP, $15 : IS IT A (CR)?
BNE 16% : BRANCH IF NO

MOV $20, SP : SAVE NEW SWR
MOV $0, SP : CLEAR UP STACK

CMP $1TAG, $1 : RE-ENABLE TTY KBD INTERP-ETE?
BNE 19% : BRANCH IF NOT

MOV $100, $3TKS : RE-ENABLE TTY KBD INTERP-ETE

RTI

JSR PC, $TYPEC : ECHO CHAR

CMP $SP, $60 : CHAR < 0?
BLT 18% : BRANCH IF YES

CMP $SP, $67 : CHAR = ?

BEQ 17% : BRANCH IF YES

BGT 18% : BRANCH IF YES

BIC $60, (SP); : STRIP-OFF ASCII

BIC $60, (SP); : IS THIS THE FIRST CHAR

BEQ 17% : BRANCH IF YES

ASL (SP) : NO, SHIFT PRESENT

ASL (SP) : CHAR OVER TO MAKE

ASL (SP) : ROOM FOR NEW ONE.

ASL (SP) : KEEP COUNT OF CHAR

ASL (SP) : SET IN NEW CHAR

BR 7% : TYPE = (CR)<(LF)

AND $SP, $712 : TYPE = (CR)X(LF)

AND $SP, $712 : SIMULATE CONTROL-U

; THIS ROUTINE WILL INPJT A SINGLE CHARACTER FROM THE TTY
; CALL: RDCHR
; RETURN HERE

RDCHR: MOV $0, (SP) : PUSH DOWN THE PC

MOV $1, (SP) : SAVE THE PS

MOV $6, (SP) : WAIT FOR

MOV $6, (SP) : A CHARACTER
40000004 MOV 0STKB,4(SP) READ THE TTY
00000004 GET RID OF JUNK IF ANY
00000023 IS IT A CONTROL-Q?
00000000 BRANCH IF NO
00010000 WAIT FOR A CHARACTER
00000002 LOOP UNTIL IT'S THERE
00000000 MAKE IT 7-BIT ASCII
00000001 IS IT A CONTROL-Q?
00000000 IF NOT DISCARD IT
00000001 YES, RESUME
00000000 IS IT UPPER CASE?
00000001 BRANCH IF YES
00000001 BRANCH IF YES
00000000 MADE IT UPPER CASE?
00000001 GO BACK TO USER
00000000 INPUT A STRING FROM THE TTY
00000001 CALL:
00000000 RETURN HERE
00000000 ADDRESS OF FIRST CHARACTER WILL BE IN THE STAC.
00000000 TERMINATOR WILL BE A BYTE OF ALL C'S
00010002 $ROLIN: MOV R3,-(SP,
00000000 $ SAVE R3
00000000 $ GET ADDRESS
00000000 $ BUFFER FULL?
00000000 BR IF YES?
00000000 RDCHAR GET CHARACTER
00000000 MOV (SP)+,(R3)
00000000 IS IT A RubOUT?
00000000 BNE 35
00000000 CLEAR THE BUFFER AND LOOP
00000000 ECHO THE CHARACTER
00000000 TYPE 93
00000000 CHECK FOR RETURN
00000000 LOOP IF NOT RETURN
00000000 CLEAR RETURN THE 15
00000000 TYPE A LINE FEED
00000000 RESTORE R3
00000000 ADJUST THE STACK AND PUT ADDRESS OF THE
00000000 FIRST ASCII CHARACTER ON IT
00000000 RETURN
00000000 STORAGE FOR ASCII CHAR. TO TYPE
00000000 TERMINATOR
00000000 RESERVE 8 BYTES FOR TTY INPUT
00000000 CONTROL "I"
00000000 CONTROL "&"
**TYPE ROUTINE**

**ROUTINE TO TYPE ASCII MESSAGE. MESSAGE MUST TERMINATE WITH A 0 BYTE.**
**THE ROUTINE WILL ADD A NUMBER OF NULL CHARACTERS AFTER A LINE FEED.**
**NULL CONTAINS THE CHARACTER TO BE USED AS THE FILLER CHARACTER.**
**NOTES:**
**FILLS CONTAINS THE NUMBER OF FILLER CHARACTERS REQUIRES.**
**FILLC CONTAINS THE CHARACTER TO FILL AFTER.**

**CALL:**

1) USING A TRAP INSTRUCTION

2) TYPE ,MESADR ; MESADR IS FIRST ADDRESS OF AN ASCII STRING

**OR**

3) TYPE ,MESADR

**MESADR**

`007174 105737 001157`

`$TYPE: TSTB $TPFLG`

1) IS THERE A TERMINAL?
`BPL 1%`

2) HALT HERE IF NO TERMINAL
`BR 3%`

3) LEAVE

4) SAVE RO

5) RO .=(SP)

6) GET ADDRESS OF ASCII STRING

7) MOV (SP),RO

8) PUSH CHARACTER TO BE TYPED ON ASCII STACK

9) IF IT ISN'T THE TERMINATOR
`BR 4%`

10) TERMINATOR POP IT OFF THE STACK

11) POP (SP)

12) ADJUST RETURN PC

13) RETURN

14) BRANCH IF <HT>

15) BRANCH IF NOT \(\text{CRLF}\)

16) \(\text{CRLF},(SP)\)

17) \(\text{TPH},(SP)\)

18) \(\text{POP} \text{(CR,CRLF) EQUIV} \text{POP} \text{(CR,(LF) EQUIV} \text{POP} \text{CRA AND LF}

19) \(\text{CHARCN (SP)}

20) BR 2%`

21) GET NEXT CHARACTER

22) OR TYPE THIS CHARACTER

23) OR IT TIME FOR FILLER CHAR?
`JSR $PC, TYPE`...

24) IF NO GO TO NEXT CHAR.

25) \(\text{NULL, (SP)}\)

26) GET # OF FILLER CHAR. NEEDED

27) AND THE NULL CHAR.

28) \(\text{DECB (SP)}\)

29) \(\text{BR 6%}

30) \(\text{IF NO POP THE NULL OFF OF STACK}\)

31) \(\text{DO TYPE A NULL}\)

32) \(\text{DO NOT COUNT AS A COUNT}\)

33) \(\text{LOOP}\)

HORIZONTAL TAB PROCESSOR

1) REPLACE TAB WITH SPACE

2) TYPE A SPACE

3) BRANCH IF NOT AT

4) TAB STOP

5) POP SPACE OFF STACK

6) GET NEXT CHARACTER
**SBTIL - CONVERT BINARY TO DECIMAL AND TYPE ROUTINE**

---

**TYPE ROUTINE**

```
;#TYPE: TSTB       #TYPE: TSPP
;BPL 1; #TYPE:       ;BEP 1; #TYPE:
;#WAIT UNTIL PRINTER IS READY
;#LOAD CHAR TO BE TYPED INTO DATA REG.
;#IS CHARACTER A CARRIAGE RETURN?
;#BRANCH IF NO
;#IS CHARACTER A LINE FEED?
;#BRANCH IF YES
;#COUNT THE CHARACTER
;#CHARACTER COUNT 5'S PAGE

;#TYPE: TSPP

;#CHARCNT: WORD 0
;#TYPE: RTS
```

---

**SBTIL - CONVERT BINARY TO DECIMAL AND TYPE ROUTINE**

---

**THIS ROUTINE IS USED TO CHANGE A 16-BIT BINARY NUMBER TO A 5-DIGIT SIGNED DECIMAL (ASCII) NUMBER AND TYPE IT. DEPENDING ON WHETHER THE NUMBER IS POSITIVE OR NEGATIVE A SPACE OR A MINUS SIGN WILL BE PLACED BEFORE THE FIRST DIGIT OF THE NUMBER. LEADING ZEROS WILL ALWAYS BE REPLACED WITH SPACES.**

**CALL:**

```
* MOV NUM,-(SP) : #PLT THE BINARY NUMBER ON THE STACK.
* TYPDS
```

**SBTIL:**

```
;#TYPDS:

MOV     R0,-(SP) ; #PUSH RO ON STACK
MOV     R1,-(SP) ; #PUSH R1 ON STACK
MOV     R2,-(SP) ; #PUSH R2 ON STACK
MOV     R3,-(SP) ; #PUSH R3 ON STACK
MOV     R5,-(SP) ; #PUSH R5 ON STACK
MOV     R6,-(SP) ; #SET BLANK SWITCH AND SIGN
MOV     R7,-(SP) ; #GET THE INPUT NUMBER
BPL     1# ; #BP IF INPUT IS POS.
NEG     R5 ; #MAKE THE BINARY NUMBER POS.
MOV     R0,-(SP) ; #MAKE THE ASCII NUMBER NEG.
;#ZERO THE CONSTANTS INDEX
;#SETUP THE OUTPUT POINTER
;#CLEAR THE BCD NUMBER
;#GET THE Constant
;#FORM THIS BCD DIGIT
;#BR IF DONE
;#INCREASE THE BCD DIGIT BY 1

BLT     R5 ; #INC R5
ADD     R3,R5 ; #ADD BACK THE CONSTANT
ST     R2,R5 ; #CHECK IF BCD DIGIT=0
BNE     5# ; #FALL THROUGH IF 0
;#STILL DOING LEADING D'S
 Sidebar
;#BR IF YES
;#"SO"
;#BR IF NO
;#YES-SET THE SIGN
;#MAKE THE BCD DIGIT ASCII
;#MAKE IT A SPACE IF NO ALREADY A DIGIT
```

---
MOVB R2 (R3)  
TST (R0)+ : PUT THIS CHARACTER IN THE OUTPUT BUFFER
: JUST INCREMENTING
CMP R0, #10: : CHECK THE TABLE INDEX
BGT 26: : GO DO THE NEXT DIGIT
BGT 6$: : GO TO EXIT
MOV R5, R2: : GET THE LSD
BR 6$: : GO CHANGE TO ASCII
9%: TSTB (SP)+ : WAS THE LSD THE FIRST NON-ZERO?
BPL 9$: : BR IF NO
17777 17776 MOV B -1(SP), -2(R3) : YES--SET THE SIGN FOR TYPING
9%: CLRB (R3) : SET THE TERMINATOR
MOV (SP)+, R5: : POP STACK INTO R5
MOV (SP)+, R5: : POP STACK INTO R3
MOV (SP)+, R5: : POP STACK INTO R2
MOV (SP)+, R1: : POP STACK INTO R1
MOV (SP)+, R0: : POP STACK INTO RC
TYPE $DBLK: : NOW TYPE THE NUMBER
MOV (SP)+, (SP): : ADJUST THE STACK
RTR: : RETURN TO JSEP

$DDBL: 10000.
100.
10.
10.
10.
10.
10.

$DBLK: .BLKW 4

.SBTL BINARY TO OCTAL (ASCII) AND TYPE

:imates a 16-BIT NUMBER TO A 6-DIGIT OCTAL (ASCII) NUMBER AND TYPE IT
:TYPOS--ENTER HERE TO SETUP SUPPRESS ZEROS AND NUMBER OF DIGITS TO TYPE
:CALL:
2664 MOV NUM, -(SP) : NUMBER TO BE TYPED
* TYPOS : CALL FOR TYPEOUT
* .BYTE N : N=1 TO 6 FOR NUMBER OF DIGITS TO TYPE
: BYTE M : M=1 OR 0
: 4=TYPE LEADING ZEROS
: 0=suppress leading zeros

:TYPOS----ENTER HERE TO TYPE OUT WITH THE SAME PARAMETERS AS "HE LAB"
:TYPOS OR TYPDC
:CALL:
2662 MOV NUM, -(SP) : NUMBER TO BE TYPED
* TYPON : CALL FOR TYPEOUT
* 
:*TYPDC----ENTER HERE FOR TYPEOUT OF A 16 BIT NUMBER
:*CALL:
2663 MOV NUM, -(SP) : NUMBER TO BE TYPED
:* TYPOC : CALL FOR TYPEOUT
* 
:TYPOS: MOV @ (SP), -(SP) : PICKUP THE MODE
2665 MOVSP, #0: : LOAD ZERO FILL SWITCH
MOV R(S), @ (SP)+ $MODE+1 : NUMBER OF DIGITS TO TYPE
RQ R2, (SP) : ADJUST RETURN ADDRESS
BR $TYPON
BINARY TO OCTAL (ASCII) AND TYPE

CODE:

$TPLOC: MOV B, #OFIL$H
SET THE ZERO FILL SWITCH

$TYPON: MOV R, #TCNT
SET THE ITERATION COUNT

MOV R3, -(SP)
SAVE R3

MOV R4, -(SP)
SAVE R4

MOV R5, -(SP)
SAVE R5

MOV R6, #MODE+1,R4
GET THE NUMBER OF DIGITS TO TYPE

SUBTRACT IT FOR MAX. ALLOWED
SAVE IT FOR USE

MOV R4, #MODE,R4
GET THE ZERO FILL SWITCH

MOV 0,(SP),R5
Pickup the Input Number

CLR R3 CLEAR THE OUTPUT WORD

ROTATE MSB INTO "C"

GO DO MSB

FORM THIS DIGIT

GET LSB OF THIS DIGIT

TYPE THIS DIGIT?

BR IF NO

BR IF YES

BR IF JUNK

TEST FOR 0

SUPPRESS THIS 0?

BR IF YES

DON'T SUPPRESS ANYMORE 3'S

MAKE THIS DIGIT ASCII?

MAKE ASCII IF NOT ALREADY

SAVE FOR TYPING

COUNT BY 1

BR IF MORE TO DO

BR IF DONE

INSURE LAST DIGIT ISN'T A BLANK

GO DO THE LAST DIGIT

MOV (SP)+, R5
RESTORE R5

MOV (SP)+, R4
RESTORE R4

MOV (SP)+, R3

MOV 2,(SP),4,(SP)
SET THE STACK FOR RETURNING

RETURN

STORE FOR ASCII DIGIT

TERMINATOR FOR TYPE ROUTINE

OCTAL DIGIT COUNTER

NUMBER OF DIGITS TO TYPE

.SBTTL TRAP DECODER

; THIS ROUTINE WILL PICKUP THE LOWER BYTE OF THE "TRAP" INSTRUCTION
; AND USE IT TO INDEX THROUGH THE TRAP TABLE FOR THE STARTING ADDRESS
; OF THE DESIRED ROUTINE. THEN USING THE ADDRESS OBTAINED IT WILL
; GO TO THAT ROUTINE.
TRAP DECORDER

$TRAP: MOV RO,-(SP) ;SAVE RO
MOV 2(SP),RO ;GET TRAP ADDRESS
TST -(RO) ;BACKUP BY 2
MOVB (RO),RO ;GET RIGHT BYTE OF TRAP
ASL RO ;POSITION FOR INDEXING
MOV $STRAD(RO),RO ;INDEX TO TABLE
RTS RO ;GO TO ROUTINE

::THIS IS USE TO HANDLE THE "GETPRI" MACRO

$TRAP2: MOV (SP),-(SP) ;MOVE THE PC DOWN
MOV 4(SP),2(SP) ;MOVE THE PSW DOWN
RTI ;RESTORE THE PSW

.SBTTL TRAP TABLE

::THIS TABLE CONTAINS THE STARTING ADDRESSES OF THE ROUTINES CALLED
::BY THE "TRAP" INSTRUCTION.

ROUTINE

$TRAP: WORD $TRAP2

.TYPE ;CALL=TYPE TRAP+1(104401) TTY TYPEOUT ROUTINE
.TYPOC ;CALL=TYPOC TRAP+2(104402) TYPE OCTAL NUMBER (WITH LEADING ZEROS)
.TYPOS ;CALL=TYPOS TRAP+3(104403) TYPE OCTAL NUMBER (NO LEADING ZEROS)
.TYPOD ;CALL=TYPOD TRAP+4(104404) TYPE OCTAL NUMBER (AS PER LAST CALL)
.TYPS ;CALL=TYPS TRAP+5(104405) TYPE DECIMAL NUMBER (WITH SIGN)

.GTSW ;CALL=GTSW TRAP+6(104406) GET SOFT-SWR SETTING

.CN.RST ;CALL=CN.RST TRAP+12(104412) CONTROL RESET ROUTINE

.DELAY ;CALL=DELAY TRAP+14(104414) TIME DELAY ROUTINE

.SBTL POWER DOWN AND UP ROUTINES

.POWER DOWN ROUTINE

$PWRDN: MOV #31LUP,2#PWRVEC ;SET FOR FAST UP
MOV #340,#PWRVEC+2 ;PWR:7
MOV RO,-(SP) ;PUSH RO ON STACK
MOV 1,(SP) ;PUSH RI ON STACK
MOV R3,-(SP) ;PUSH RE ON STACK
MOV R4,-(SP) ;PUSH RE ON STACK
MOV R5,-(SP) ;PUSH RE ON STACK
MOV $SWR,-(SP) ;PUSH $SWR ON STACK
MOV SP,$SWR6 ;SAVE SP
<table>
<thead>
<tr>
<th>EM</th>
<th>Address</th>
<th>Value</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>010474</td>
<td>000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>010502</td>
<td>044516</td>
<td>020124</td>
<td>044504</td>
</tr>
<tr>
<td>010510</td>
<td>020104</td>
<td>047516</td>
<td>020124</td>
</tr>
<tr>
<td>010512</td>
<td>046103</td>
<td>040506</td>
<td>020122</td>
</tr>
<tr>
<td>010524</td>
<td>045622</td>
<td>051503</td>
<td>000</td>
</tr>
<tr>
<td>010531</td>
<td>020116</td>
<td>046122</td>
<td>061228</td>
</tr>
<tr>
<td>010544</td>
<td>042040</td>
<td>042111</td>
<td>023516</td>
</tr>
<tr>
<td>010550</td>
<td>020122</td>
<td>046103</td>
<td>040506</td>
</tr>
<tr>
<td>010560</td>
<td>020122</td>
<td>045522</td>
<td>051503</td>
</tr>
<tr>
<td>010560</td>
<td>020054</td>
<td>047111</td>
<td>051440</td>
</tr>
<tr>
<td>010572</td>
<td>052106</td>
<td>047111</td>
<td>020107</td>
</tr>
<tr>
<td>010602</td>
<td>043447</td>
<td>023517</td>
<td>000</td>
</tr>
<tr>
<td>010607</td>
<td>020116</td>
<td>046122</td>
<td>061228</td>
</tr>
<tr>
<td>010614</td>
<td>051040</td>
<td>051505</td>
<td>052105</td>
</tr>
<tr>
<td>010624</td>
<td>042111</td>
<td>023516</td>
<td>020124</td>
</tr>
<tr>
<td>010630</td>
<td>042523</td>
<td>020124</td>
<td>043101</td>
</tr>
<tr>
<td>010636</td>
<td>042523</td>
<td>020122</td>
<td>043103</td>
</tr>
<tr>
<td>010644</td>
<td>051124</td>
<td>020114</td>
<td>042522</td>
</tr>
<tr>
<td>010652</td>
<td>042523</td>
<td>000124</td>
<td></td>
</tr>
<tr>
<td>010656</td>
<td>042522</td>
<td>044507</td>
<td>052123</td>
</tr>
<tr>
<td>010664</td>
<td>051105</td>
<td>047040</td>
<td>052117</td>
</tr>
<tr>
<td>010672</td>
<td>041440</td>
<td>042514</td>
<td>051101</td>
</tr>
<tr>
<td>010700</td>
<td>042105</td>
<td>000</td>
<td></td>
</tr>
<tr>
<td>010703</td>
<td>052513</td>
<td>020103</td>
<td>011122</td>
</tr>
<tr>
<td>010710</td>
<td>051105</td>
<td>047522</td>
<td>000122</td>
</tr>
<tr>
<td>010716</td>
<td>045522</td>
<td>040502</td>
<td>042440</td>
</tr>
<tr>
<td>010724</td>
<td>051122</td>
<td>051117</td>
<td>000</td>
</tr>
<tr>
<td>010731</td>
<td>052116</td>
<td>046122</td>
<td>061228</td>
</tr>
<tr>
<td>010736</td>
<td>051040</td>
<td>051505</td>
<td>052105</td>
</tr>
<tr>
<td>010744</td>
<td>042040</td>
<td>042111</td>
<td>023516</td>
</tr>
<tr>
<td>010752</td>
<td>020124</td>
<td>046103</td>
<td>040506</td>
</tr>
<tr>
<td>010760</td>
<td>020122</td>
<td>042622</td>
<td>04507</td>
</tr>
<tr>
<td>010766</td>
<td>052123</td>
<td>051105</td>
<td>000</td>
</tr>
<tr>
<td>010773</td>
<td>022</td>
<td>042113</td>
<td>020101</td>
</tr>
<tr>
<td>010800</td>
<td>051105</td>
<td>047522</td>
<td>000122</td>
</tr>
<tr>
<td>010906</td>
<td>052602</td>
<td>020123</td>
<td>047111</td>
</tr>
<tr>
<td>010914</td>
<td>052111</td>
<td>042040</td>
<td>042111</td>
</tr>
<tr>
<td>010922</td>
<td>043516</td>
<td>020124</td>
<td>046103</td>
</tr>
<tr>
<td>010930</td>
<td>020122</td>
<td>042622</td>
<td>04507</td>
</tr>
<tr>
<td>010936</td>
<td>020123</td>
<td>000126</td>
<td>000</td>
</tr>
<tr>
<td>010942</td>
<td>042101</td>
<td>051104</td>
<td>051505</td>
</tr>
<tr>
<td>010950</td>
<td>045522</td>
<td>043516</td>
<td>042440</td>
</tr>
<tr>
<td>010956</td>
<td>051122</td>
<td>051117</td>
<td>052155</td>
</tr>
<tr>
<td>Error Code</td>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EM22</td>
<td>.ASCIZ 'DIDN'T CLEAR RKCS LOW BYTE'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EM23</td>
<td>.ASCIZ 'DIDN'T CLEAR RKCS HIGH BYTE'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EM24</td>
<td>.ASCIZ 'TRIED TO CLEAR RKCS 'BYTE', CHANGED REGIS'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EM25</td>
<td>.ASCIZ 'FAILED TO CLEAR 'REG-BYTE'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EM26</td>
<td>.ASCIZ 'RKCS ALTERED ON CLEARING 'REG-BYTE'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EM27</td>
<td>.ASCIZ 'TRIED TO CLEAR 'REG-BYTE', CHANGED 'REG-BYTE'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EM43</td>
<td>.ASCIZ 'UNEXPECTED RKII INTERRUPT'</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
F05

MACYII 30.10M6 06-JUN-77 14:40 PAGE 5

ERROR HEADERS

3006 011770 051050 043505 024461
3007 011778 020040 051050 043505
3008 012009 024462 000
3009
30010 012007 040 050040 020103 DH24: .ASCIZ PC BYTE REGS (REG, EXP, REG, RECVD)
30011 012014 020040 020040 054502
30012 012022 042524 020040 051040
30013 012030 043505 051511 024040
30014 012036 041522 024507 054105
30015 012044 020120 051050 043505
30016 012052 051051 041505 042126
30017 012060 000
30018 012061 040 050040 020103 DH25: .ASCIZ PC REG-BYTE RECVD
30019 012064 020040 020040 056507
30020 012074 041505 042126 051040
30021 012076 041505 042126 000
30022 012107 040 050040 020103 DH26: .ASCIZ PC REG-BYTE (CS, EXP, CS, RECVD)
30023 012114 020040 020120 056507
30024 012122 042522 020120 054105
30025 012130 041503 042451 051030
30026 012136 042400 051503 051051
30027 012144 041505 042126 000
30028 012151 040 050040 020103 DH27: .ASCIZ PC R-BYTE1 R-BYTE2 2-EXPCT 2-RECVD
30029 012156 020040 051040 041056
30030 012164 052131 020061 051040
30031 012172 051055 052131 020066
30032 012190 031040 042452 051030
30033 012196 052131 031040 051055
30034 012214 041505 042126 000
30035 012221 040 050040 020103 DH3C: .ASCIZ PC ACKS ACKE ACKS
30036 012226 020040 020040 056502
30037 012224 020040 020040 020040
30038 012232 041503 051105 020040
30039 012240 020040 042452 051030
30040 012246 030 045522 051504

...END...