ML11 PROM MAINT PROG CZMLCB0 AH-S511B-MC FICHE 1 OF 2 JUL 1982
COPYRIGHT© 81-82
MADE IN USA

ML11 PROM MAINT PROG CZMLCB0 AH-S511B-MC FICHE 2 OF 2

JUL 1982
COPYRIGHT© 81-82
MADE IN USA

```
MODULE BSKEL1 =
%TITLE  'CZMLCB0 ML-11 FROM MAINTENANCE PROGRAM'                    SEQ 0001

%(
                IDENTIFICATION
                --------------

PRODUCT CODE:   AC-S509B-MC

PRODUCT NAME:   CZMLCB0 ML-11 PROM MAINTENANCE PROGRAM

PRODUCT DATE:   19-MAR-82

MAINTAINER:     MEMORY DIAGNOSTICS ENGINEERING

AUTHOR:         D.W. NEALE
```

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 DOCUMENT.

NO RESPONSIBILITY IS ASSUMED FOR THE USE OR RELIABILITY OF SOFTWARE ON EQUIPMENT THAT IS NOT SUPPLIED BY DIGITAL OR ITS AFFILIATED COMPANIES.

COPYRIGHT (C) 1981, 1982 BY DIGITAL EQUIPMENT CORPORATION

THE FOLLOWING ARE TRADEMARKS OF DIGITAL EQUIPMENT CORPORATION:

DIGITAL	PDP	UNIBUS	MASSBUS
DEC	DECUS	DECTAFE	

TABLE OF CONTENTS

1.0 GENERAL INFORMATION

1.1 PROGRAM ABSTRACT

THE ML-11 MEMORY SYSTEM WITH ITS MOSTLY ARRAY TECHNOLOGY HAS THE FACILITY TO OFFSET AROUND KNOWN BAD MEMORY LOCATIONS IN ITS MEMORY ARRAYS.

INITIALLY, THESE MEMORY ARRAYS ARE TESTED FOR BAD ROW AND COLUMN ADDRESS LOCATIONS AND THE SPECIFIC OFFSETTING INFORMATION IS STORED IN PROM ON THE ARRAY MODULE. THIS TESTING IS DONE ON A SPECIAL 2224 MEMORY TESTER BY MEMORY MANUFACTURING.

THE DESIGN OF THE ML11 HAS ALSO PROVIDED LOGIC THAT WHEN UNDER SOFTWARE CONTROL WILL UPDATE AN ARRAY MODULES OFFSETTING INFORMATION. THIS LOGIC IS TO BE UTILIZED WHEN ADDITIONAL MEMORY CELLS ARE DISCOVERED BAD AFTER THE SYSTEM HAS LEFT THE MANUFACTURING FACILITY.

FIELD SERVICE HAS REQUESTED THE CREATION OF A PROGRAM THAT WHEN RUN WILL TEST A GIVEN ML-11A OR ML-11B SYSTEM FOR ANY ADDITIONAL BAD MEMORY CELLS AND UPDATE THE OFFSET INFORMATION SUCH THAT THESE BAD LOCATIONS WILL BE MASKED FROM FURTHER OCCURANCES.

THIS PROGRAM WILL SELECTIVELY UPDATE BAD MEMORY CELL OFFSETING FOR AN ENTIRE ML-11 SYSTEM, A SINGLE ARRAY MODULE OR A SINGLE BANK.

THE PROGRAM WILL EXERCISE AN ML-11A OR ML-11B WITH ALL ONES, ALL ZEROES AND RANDOM DATA PATTERNS TO FIND ANY ADDITIONAL FAILING MEMORY CELLS.

ONCE THE ADDITIONAL FAILING CELLS HAVE BEEN MASKED OUT THE PROGRAM WILL GO BACK INTO THE FAILING CELLS AND VERIFY THAT THESE BAD CELLS HAVE INDEED BEEN MASKED OUT.

THE OPERATOR WILL BE NOTIFIED OF ANY ERROR CONDITIONS WHICH MIGHT OCCURE DURING THE EXECUTION OF THE PROGRAM. ADDITIONAL INFORMATION PERTAINING TO THESE ERROR MESSAGES CAN BE FOUND IN SECTION 3.0 OF THIS DOCUMENT.

THIS DIAGNOSTIC HAS BEEN WRITTEN FOR USE WITH THE DIAGNOSTIC RUNTIME SERVICES SOFTWARE (DRS>). THESE SERVICES PROVIDE THE INTERFACE TO THE OPERATOR AND TO THE SOFTWARE ENVIRONMENT.

THIS PROGRAM CAN BE USED WITH XXDP+, ACT, APT, SLIDE. FOR A COMPLETE DESCRIPTION OF THE RUNTIME SERVICES, REFER TO THE XXDP+ USER'S MANUAL. THERE IS A BRIEF DESCRIPTION OF THE RUNTIME SERVICES IN SECTION 2 OF THIS DOCUMENT.

1.2 SYSTEM REQUIREMENTS

1. PDP-11 CENTRAL PROCESSOR WITH A MINIMUM OF 28K USABLE MAIN MEMORY.

2. CONSOLE TERMINAL.

3. RH11 OR RH70 DISK CONTROLLER.

4. A MINIMUM OF ONE ML-11A AND ML-11B SYSTEM ATTACHED TO THE E 1 ABOVE RH CONTROLLER.

5. XXDP+ LOAD MEDIA.

1.3 RELATED DOCUMENTS AND STANDARDS

1. SUPPRGC.DOC
2. SUPINT.MEM
3. SUPFUN.C
4. XXDPPLUS.DOC
5. BLISS LANGUAGE GUIDE
6. BLISS-16 USER'S GUIDE

1.4 ASSUMPTIONS

IT WILL BE ASSUMED THAT PRIOR TO RUNNING OF THIS PROGRAM THAT ALL APPROPRIATE CPU AND MAIN MEMORY DIAGNOSTICS HAVE BEEN SUCCESSFULLY RUN ON THE ML-11'S HOST SYSTEM.

IT IS FURTHER ASSUMED THAT THE ML-11 LOGIC TEST AND THE ML-11 SYSTEM EXERCISER HAS BEEN SUCCESSFULLY RUN ON THE ML-11 SYSTEM AND THAT THE SYSTEM EXERCISER HAS SPECIFICALLY CALLED OUT THE RUNNING OF THE PROM MAINTENANCE ON THIS UNIT.

THE SYSTEM EXERCISER WHEN IT CALLS FOR THE RUNNING OF THIS PROGRAM WILL INDICATE WHICH ARRAYS AND BANKS OF THE ML-11 SYSTEM NEED TO BE PROM MAINTENANCED. THIS INFORMATION SHOULD BE THEN INPUTED INTO THIS PROGRAM.

HOWEVER THIS PROGRAM IS DESIGNED TO GIVE THE OPERATOR THE OPTION TO PROM MAINTENANCE EITHER THE ARRAY AND BANK THAT THE SYSTEM EXERCISER CALLS OUT FOR PM'ING OR SELECT PROM MAINTENANCE FOR AN ENTIRE ARRAY MODULE (FOUR BANKS) OR SELECT PROM MAINTENANCE FOR THE ENTIRE ML-11 SYSTEM (ALL PRESENT ARRAY MODULES).

2.0 OPERATING INSTRUCTIONS

THIS SECTION CONTAINS A BRIEF DESCRIPTION OF THE RUNTIME SERVICES. FOR DETAILED INFORMATION, REFER TO THE XXDP+ USER'S MANUAL (CHQUS).

2.1 COMMANDS

THERE ARE ELEVEN LEGAL COMMANDS FOR THE DIAGNOSTIC RUNTIME SERVICES (SUPERVISOR). THIS SECTION LISTS THE COMMANDS AND GIVES A VERY BRIEF DESCRIPTION OF THEM. THE XXDP+ USER'S MANUAL HAS MORE DETAILS.

COMMAND	EFFECT

START	START THE DIAGNOSTIC FROM AN INITIAL STATE
RESTART	START THE DIAGNOSTIC WITHOUT INITIALIZING
CONTINUE	CONTINUE AT TEST THAT WAS INTERRUPTED (AFTER ^C)
PROCEED	CONTINUE FROM AN ERROR HALT
EXIT	RETURN TO XXDP+ MONITOR (XXDP+ OPERATION ONLY!)
ADD	ACTIVATE A UNIT FOR TESTING (ALL UNITS ARE CONSIDERED TO BE ACTIVE AT START TIME
DROP	DEACTIVATE A UNIT
PRINT	PRINT STATISTICAL INFORMATION (IF IMPLEMENTED BY THE DIAGNOSTIC - SECTION 4.0)
DISPLAY	TYPE A LIST OF ALL DEVICE INFORMATION
FLAGS	TYPE THE STATE OF ALL FLAGS (SEE SECTION 2.3)
ZFLAGS	CLEAR ALL FLAGS (SEE SECTION 2.3)

A COMMAND CAN BE RECOGNIZED BY THE FIRST THREE CHARACTERS. SO YOU MAY, FOR EXAMPLE, TYPE "STA" INSTEAD OF "START".

THIS PROGRAM USES THE DIAGNOSTIC RUN TIME SERVICE FOR PROGRAM PARAMETER INPUT, ERROR REPORTING AND MESSAGE PRINTING.

IT IS DESINGED TO TEST ONE ML-11 SYSTEM AND IS EXPECTED TO RUN FROM START TO FINISH WITH NO OPERATOR INTERRUPTIONS (ie. ^C).

THEREFORE THE ONLY RECOGNIZED DRS> COMMAND BY THIS PROGRAM IS THE 'START' COMMAND AND CONTROL C AND ANY OTHER DRS> COMMAND MUST BE AVOIDED.

FOR STATISTICAL ANALYSIS THIS PROGRAM, UPON AN DRS> COMMAND 'PRINT', WILL DISPLAY TO THE CONSOLE TERMINAL A REPORT SUMMARY INDICATING WHERE THE PROGRAM HAS FOUND ADDITIONAL FAILING MEMORY CHIPS AND A COUNT OF FAILING ROWS AND COLUMNS WITHIN EACH CHIP.

2.2 SWITCHES

THERE ARE SEVERAL SWITCHES WHICH ARE USED TO MODIFY SUPERVISOR OPERATION. THESE SWITCHES ARE APPENDED TO THE LEGAL COMMANDS. ALL OF THE LEGAL SWITCHES ARE TABULATED BELOW WITH A BRIEF DESCRIPTION OF EACH. IN THE DESCRIPTIONS BELOW, A DECIMAL NUMBER IS DESIGNATED BY "DDDDD".

SWITCH	EFFECT
/TESTS:LIST	EXECUTE ONLY THOSE TESTS SPECIFIED IN THE LIST. LIST IS A STRING OF TEST NUMBERS, FOR EXAMPLE - /TESTS:1:5:7-10. THIS LIST WILL CAUSE TESTS 1,5,7,8,9,10 TO BE RUN. ALL OTHER TESTS WILL NOT BE RUN.
/PASS:DDDDD	EXECUTE DDDDD PASSES (DDDDD = 1 TO 64000)
/FLAGS:FLGS	SET SPECIFIED FLAGS. FLAGS ARE DESCRIBED IN SECTION 2.3.
/EOP:DDDDD	REPORT END OF PASS MESSAGE AFTER EVERY DDDDD PASSES ONLY. (DDDDD = 1 TO 64000)
/UNITS:LIST	TEST/ADD/DROP ONLY THOSE UNITS SPECIFIED IN THE LIST. LIST EXAMPLE - /UNITS:0:5:10-12 USE UNITS 0,5,10,11,12 (UNIT NUMBERS = 0-63)

EXAMPLE OF SWITCH USAGE:

```
START/TESTS:1-5/PASS:1000/EOP:100
```

THE EFFECT OF THIS COMMAND WILL BE: 1) TESTS 1 THROUGH 5 WILL BE EXECUTED, 2) ALL UNITS WILL BE TESTED 1000 TIMES AND 3) THE END OF PASS MESSAGES WILL BE PRINTED AFTER EACH 100 PASSES ONLY. A SWITCH CAN BE RECOGNIZED BY THE FIRST THREE CHARACTERS. YOU MAY, FOR EXAMPLE, TYPE "/TES:1-5" INSTEAD OF "/TESTS:1-5".

BELOW IS A TABLE THAT SPECIFIES WHICH SWITCHES CAN BE USED BY EACH COMMAND.

	TESTS	PASS	FLAGS	EOP	UNITS
START	X	X	X	X	X
RESTART	X	X	X	X	X
CONTINUE		X	X	X	
PROCEED			X		
DROP					X
ADD					X
PRINT					
DISPLAY					X
FLAGS					
ZFLAGS					
EXIT					

AS MENTIONED BEFORE THE PROGRAM IS DESIGNED TO TEST ONE ML-11 SYSTEM AND IS EXPECTED TO RUN FROM START TO FINISH WITH NO OPERATOR INTERRUPTIONS.

THEREFORE USAGE OF ANY SWITCHES WOULD PROVE MEANINGLESS TO THE PROGRAM AND SHOULD BE AVOIDED.

2.3 FLAGS

FLAGS ARE USED TO SET UP CERTAIN OPERATIONAL PARAMETERS SUCH AS LOOPING ON ERROR. ALL FLAGS ARE CLEARED AT STARTUP AND REMAIN CLEARED UNTIL EXPLICITLY SET USING THE FLAGS SWITCH. FLAGS ARE ALSO CLEARED AFTER A START COMMAND UNLESS SET USING THE FLAG SWITCH. THE ZFLAGS COMMAND MAY ALSO BE USED TO CLEAR ALL FLAGS. WITH THE EXCEPTION OF THE START AND ZFLAGS COMMANDS, NO COMMANDS AFFECT THE STATE OF THE FLAGS; THEY REMAIN SET OR CLEARED AS SPECIFIED BY THE LAST FLAG SWITCH.

FLAG	EFFECT
HOE	HALT ON ERROR - CONTROL IS RETURNED TO RUNTIME SERVICES COMMAND MODE
LOE	LOOP ON ERROR
IER*	INHIBIT ALL ERROR REPORTS
IBR*	INHIBIT ALL ERROR REPORTS EXCEPT FIRST LEVEL (FIRST LEVEL CONTAINS ERROR TYPE, NUMBER, PC, TEST AND UNIT)
IXR*	INHIBIT EXTENDED ERROR REPORTS (THOSE CALLED BY PRINTX MACRO'S)
PRI	DIRECT MESSAGES TO LINE PRINTER
PNT	PRINT TEST NUMBER AS TEST EXECUTES
BOE	"BELL" ON ERROR
UAM	UNATTENDED MODE (NO MANUAL INTERVENTION)
ISR	INHIBIT STATISTICAL REPORTS (DOES NOT

	APPLY TO DIAGNOSTICS WHICH DO NOT SUPPORT STATISTICAL REPORTING)
IDR	INHIBIT PROGRAM DROPPING OF UNITS
ADR	EXECUTE AUTODROP CODE
LOT	LOOP ON TEST
EVL	EXECUTE EVALUATION (ON DIAGNOSTICS WHICH HAVE EVALUATION SUPPORT)

*ERROR MESSAGES ARE DESCRIBED IN SECTION 3.1

SEE THE XXDP+ USER'S MANUAL FOR MORE DETAILS ON FLAGS. YOU MAY SPECIFY MORE THAN ONE FLAG WITH THE FLAG SWITCH. FOR EXAMPLE, TO CAUSE THE PROGRAM TO LOOP ON ERROR, INHIBIT ERROR REPORTS AND TYPE A "BELL" ON ERROR, YOU MAY USE THE FOLLOWING STRING:

/FLAGS:LOE:IER:BOE

2.4 HARDWARE QUESTIONS

WHEN A DIAGNOSTIC IS STARTED, THE RUNTIME SERVICES WILL PROMPT THE USER FOR HARDWARE INFORMATION BY TYPING "CHANGE HW (L) ?" YOU MUST ANSWER "Y" AFTER A START COMMAND UNLESS THE HARDWARE INFORMATION HAS BEEN "PRELOADED" USING THE SETUP UTILITY (SEE CHAPTER 6 OF THE XXDP+ USER'S MANUAL). WHEN YOU ANSWER THIS QUESTION WITH A "Y", THE RUNTIME SERVICES WILL ASK FOR THE NUMBER OF UNITS (IN DECIMAL). ONLY ONE DRIVE IS PERMITTED TO BE PROM MAINTENANCED PER EXECUTION OF THE PROGRAM THEREFOR ANSWER THIS QUESTION WITH '1'. YOU WILL THEN BE ASKED THE FOLLOWING QUESTIONS.

OPTION 1	'IS ENTIRE ML-11 SYSTEM TO BE MASKED' TRANSFER 'SYS' IF TRUE
OPTION 2	'IS A SINGLE ARRAY TO BE MASKED' TRANSFER 'BOARD' IS TRUE
OPTION 3	'IS A SINGLE BANK TO BE MASKED' TRANSFER 'DONE' IF FALSE
	'ENTER BANK NUMBER TO BE MASKED'
'BOARD'	'ENTER BOARD NUMBER TO BE MASKED'
'SYS'	'STARTING RH REGISTER ADDRESS'
	'DRIVE UNDER TEST NUMBER'
'DONE'	'ARE YOUR INPUTED PARAMETERS CORRECT'

2.5 SOFTWARE QUESTIONS

SOFTWARE QUESTIONS ARE NOT USED DURING THIS PROGRAM AND THIS QUESTION SHOULD BE ANSWERED WITH A 'NO' RESPONCE.

HOWEVER IF A YES RESPONCE IS GIVEN THE FOLLOWING MESSAGE WILL BE PRINTED:

'NOT USED TYPE <CR>'

2.6 EXTENDED P-TABLE DIALOGUE

TRADITIONALLY DRS> PROVIDES YOU WITH THE ABILITY TO BUILD P-TABLES FOR MULTIPLE DRIVE TESTING. BECAUSE OF THE IMPACT OF THIS PROGRAM ON AN ML-11 SYSTEM AND THE LENGTHY RUNTIME ONLY ONE DRIVE WILL BE PROM MAINTENANCED PER EXECUTION OF THIS PROGRAM.

HOWEVER THE NATURE OF DRS> WILL STILL ALLOW YOU TO BUILD MULTIPLE DRIVE SELECTION FOR TESTING. THIS PROGRAM WILL TREAT THIS AS A SYSTEM ERROR AND SELECT THE FIRST P-TABLE BUILT FOR THE RUN TIME PARAMETERS.

2.7 START-UP PROCEDURE (XXDP+)

TO START-UP THIS PROGRAM:

1. BOOT XXDP+
2. ENTER THE DATE
3. TYPE "R CZMLC"
4. TYPE "START"

   THE START COMMAND WILL BE THE ONLY COMMAND ACCEPTED BY THIS PROGRAM AND TYPING ANY OTHER COMMAND MUST BE AVOIDED. FOR TESTING MULTIPLE DRIVES REPEAT STEPS 4 THRU 7 FOR EACH DRIVE.

5. ANSWER THE "CHANGE HW" QUESTION WITH "Y"
6. ANSWER ALL THE HARDWARE QUESTIONS
7. ANSWER THE "CHANGE SW" QUESTION WITH "N"

WHEN YOU FOLLOW THIS PROCEDURE YOU WILL BE USING ONLY THE DEFAULTS FOR FLAGS AND SOFTWARE PARAMETERS. THESE DEFAULTS ARE DESCRIBED IN SECTIONS 2.3 AND 2.5.

3.0 ERROR INFORMATION

3.1 TYPES OF ERROR MESSAGES

THERE ARE THREE LEVELS OF ERROR MESSAGES THAT MAY BE ISSUED BY A DIAGNOSTIC: GENERAL, BASIC AND EXTENDED. GENERAL ERROR MESSAGES ARE ALWAYS PRINTED UNLESS THE "IER" FLAG IS SET (SECTION 2.3). THE GENERAL ERROR MESSAGE IS OF THE FORM:

```
NAME  TYPE  NUMBER  ON UNIT NUMBER  TST NUMBER PC:XXXXXX
ERROR MESSAGE
```

```
,WHERE; NAME = DIAGNOSTIC NAME
        TYPE = ERROR TYPE (SYS FATAL, DEV FATAL, HARD OR SOFT)
        NUMBER = ERROR NUMBER
        UNIT NUMBER = 0 - N (N IS LAST UNIT IN PTABLE)
        TST NUMBER = TEST AND SUBTEST WHERE ERROR OCCURRED
```

PC:XXXXXX = ADDRESS OF ERROR MESSAGE CALL

BASIC ERROR MESSAGES ARE MESSAGES THAT CONTAIN SOME ADDITIONAL INFORMATION ABOUT THE ERROR. THESE ARE ALWAYS PRINTED UNLESS THE "IER" OR "IBR" FLAGS ARE SET (SECTION 2.3). THESE MESSAGES ARE PRINTED AFTER THE ASSOCIATED GENERAL MESSAGE.

EXTENDED ERROR MESSAGES CONTAIN SUPPLEMENTARY ERROR INFORMATION SUCH AS REGISTER CONTENTS OR GOOD/BAD DATA. THESE ARE ALWAYS PRINTED UNLESS THE "IER", "IBR" OR "IXR" FLAGS ARE SET (SECTION 2.3). THESE MESSAGES ARE PRINTED AFTER THE ASSOCIATED GENERAL ERROR MESSAGE AND ANY ASSOCIATED BASIC ERROR MESSAGES.

3.2 SPECIFIC ERROR MESSAGES

ERROR NUMBER	ERROR DESCRIPTION
ERR_1	ONLY THE DRS> START COMMAND IS RECOGNIZED TO START THE PROGRAM EXECUTION. ANY OTHER DRS> COMMAND WILL CAUSE THIS ERROR
ERR_2	DURING THE HARDWARE QUESTIONS THE OPERATOR IS ASKED IF A SINGLE BANK, A SINGLE ARRAY OR THE ENTIRE ML-11 SYSTEM IS TO BE PROM MAINTENANCED. HE/SHE IS THEN ASKED IF HIS/HER INPUTS ARE CORRECT. THIS ERROR DETECTS A NO ANSWER FOR SELECTING SYSTEM, ARRAY OR BANK AND A YES ANSWER TO 'ARE YOUR INPUTED PARAMETERS CORRECT'.
ERR_3	EVEN THOUGH DRS> WILL BUILD MULTIPLE P-TABLES THIS PROGRAM WILL USE AS ITS RUN TIME PARAMETERS THE LUN 0'S ENTRIES. THIS ERROR DETECTS THE ABSENCE OF THIS FIRST P-TABLE 'LUN 0'.
ERR_4	THIS ERROR DETECTS UNCONFIRMED FAILING CHIPS. ie. CHIPS THAT FAILED DURING MASS BUS WRITE CHECK TRANSFERS BUT THE FAILURE DID NOT REOCCUR DURING DATA DIAGNOSTIC MODES. THIS MAY INDICATE THAT THIS FAILURE MAY BE A SOFT ERROR OR POSSIBLE HARDWARE ERRORS.
ERR_5	CONDITION A THIS INDICATES THAT 'ONE' ALL BAD CHIP (GREATER THAN 10 ALL BAD ROWS AND OR COLUMNS) HAS BEEN DETECTED IN A CHIP AT A GIVEN BANK. THIS CHIP IS NOT PROM MAINTENANCED AND THE ERROR CORRECTION IS EXPECTED TO CORRECT THE FAILING DATA

FROM THIS CHIP UNTIL FIELD SERVICE CAN REPLACE
THE ARRAY FROM WHICH THIS CHIP RESIDES.

ERR_6 CONDITION B

THIS INDICATES THAT A SECOND ALL BAD CHIP HAS
BEEN DETECTED IN A GIVEN BANK.

THE BAD CHIP IS NOT PROM MAINTENANCED AND FURTHER
TESTING OF THIS ARRAY IS ABORTED.

FIELD SERVICE SHOULD REPLACE THIS ARRAY BEFORE
LEAVING THE SITE.

ERR_7 CONDITION C

INDICATES THAT BAD NIBBLE OFFSETS HAVE EXCEEDED
14 OFFSETS RESULTING IN UNSAFE ERRORS.

THE ARRAYS NIBBLE OFFSETS FOR THIS BANK ARE MASKED
UP TO 14 OFFSETS. THE ERROR CORRECTION IS EXPECTED
TO CORRECT THE UNMASKED ERRORS WHICH WERE LEFT
BEHIND.

FIELD SERVICE SHOULD REPLACE THIS ARRAY MODULE
AT THE EARLIEST POSSIBLE DATA.

ERR_8 CONDITION D

THIS INDICATES THAT PROM BLAST ERRORS WERE DETECTED.

THIS MEANS THAT THE SELECTED NEW PROM DATA WAS NOT
WRITTEN INTO THE ARRAYS PROMS CORRECTLY. EITHER THE
SELECTED PROM LOCATIONS WERE NOT WRITTEN CORRECTLY
OR OTHER PROM LOCATIONS WERE INADVERTENTLY WRITTEN INTO.

THIS COULD RESULT IN EITHER BAD LOCATIONS
ARE NOT BEING MASKED OR GOOD LOCATIONS ARE BEING
MASKED.

LATER ROUTINES WILL DETERMINE IF THIS SITUATION
WARRENTS THE ARRAY TO BE REPLACED.

IF NO OTHER ERROR MESSAGES OCCUR AFTER CONDITION D
MESSAGES THEN IGNORE THIS ERROR.

ERR_9 AS MENTIONED BEFORE ONLY THE FIRST P-TABLE BUILT
WILL BE USED AS THE PROGRAM PARAMETERS.

THIS ERROR DETECTS THAT MORE THAN ONE P-TABLE
WAS BUILT DURING THE HARDWARE QUESTIONS.

ERR_10 DURING WRITING NEW PROM DATA TO THE PROMS THE
DATA CLOCK BIT LOCATED ON THE ARRAY DATA MODULE
IS TESTED FOR COMPLETION OF THE PROM WRITE.

THIS ERROR DETECTS THE FAILURE OF THIS BIT TO
CLEAR AFTER WRITING TO THE PROMS.

ERR_11 AFTER THE PROMS HAVE BEEN WRITTEN WITH NEW PROM

DATA THE PROGRAM VERIFIES THAT ALL NEWLY FAILING
ROWS AND COLUMNS HAVE BEEN SUCCESSFULLY MASKED OUT.

THIS ERROR DETECTS THE OCCURANCE OF UNCORRECTABLE
ERROR DURING THIS VERIFY PASS.

THIS ERROR CAN NOT BE TOLERATED AND THE ARRAY IS
CALLED OUT FOR REPLACEMENT.

FIELD SERVICE MUST REPLACE THIS ARRAY MODULE
BEFORE LEAVING THE SITE.

IF THIS ERROR STILL EXITS AFTER THE ARRAY IS
REPLACED THEN POSSIBLE HARDWARE ERRORS MAY
EXITS IN THE DRIVE. THE LOGIC TEST AND EXERCISER
SHOULD BE RUN AGAIN.

ERR_12 AGAIN DURING THE VERIFY PASS THE DRIVE IS EXAMINED
FOR ERRORS AFTER BLASTING.

SINGLE BIT ERRORS ARE TOLERATED IN BANKS WHICH
ARE RUNNING DEGRADE MODE (EITHER ONE ALL BAD CHIP
WAS LEFT BEHIND OR NIBBLE OFFSETS GREATER THAN
WERE DETECTED). HOWEVER NON - DEGRADE MODE BANKS
SHOULD BE RUNNING ERROR FREE AFTER BLASTING.

THIS ERROR DETECTS THE OCCURANCE OF ERRORS
IN THE BANKS WHICH ARE NOT IN DEGRADE MODE.

THIS COULD INDICATE THAT THE PROGRAM FAILED TO
TO FIND AND MASK OUT ADDITIONAL ERRORS IN THE BANK
OR THAT POSSIBLE HARDWARE ERRORS EXIST.

THE PROGRAM SHOULD BE RUN ON THIS BANK AGAIN.
IF THE ERROR STILL EXISTS THEN ISOLATE WHERE
THE PROBLEM LIES. IF THIS PROGRAM IS SUSPECTED
THEN CONTACT MEMORY DIAGNOSTIC ENGINEERING.

ERR_13 THIS ERROR DETECTS THE PRESENTS OF UNEXPECTED
DRIVE ERRORS DURING OR AFTER A MASS BUS TRANS-
FER. AFTER THE A MESSAGE IS PRINTED STATING THE ERROR
ALL THE DIRECTLY READ ML-11 REGISTERS ARE DUMPED
TO THE TERMINAL FOR OPERATOR REVIEW.

4.0 PERFORMANCE AND PROGRESS REPORTS

AT THE END OF EACH PASS, THE PASS COUNT IS GIVEN ALONG WITH THE
TOTAL NUMBER OF ERRORS REPORTED SINCE THE DIAGNOSTIC WAS STARTED.

THIS PROGRAM WILL BE EXECUTED ONE TIME PER 'START' COMMAND ISSUED.
THE DRS> REPORT OF HOW MANY OF ERRORS DETECTED HAS NO MEANING DURING
EXECUTION OF THIS PROGRAM.

ONCE THE PROGRAM EXECUTION HAS COMPLETED THIS JUST PM'ED UNIT IS
DROPPED TO SUPPRESS FURTHER PROGRAM EXECUTION AND CONTROL IS PASSED
TO DRS>.

DROPPING OF A UNIT RESULTS IN A DRS> MESSAGE OF 'PASS ABORTED FOR

THIS UNIT'. THIS MESSAGE HAS NO SIGNIFICANCE ON THE PROGRAMS EXECUTION AND SHOULD BE IGNORED.

A REPORT OF THE PROGRAMS PERFORMANCE CAN BE OBTAINED VIA THE DRS> COMMAND 'PRINT'.

5.0 DEVICE INFORMATION TABLES

HARDWARE P-TABLE ENTRY DEFINTITION

TABLE LOCATION	DESCRIPTION
1. A_HWTBL : INITIAL(0)	STORES A TRUE OR FALSE VALUE AND HAS NO DEFAULT VALUE. SELECTS WHETHER THE ENTIRE ML-11 SYSTEM (ALL PRESENT ARRAYS) ARE TO BE PM'ED.
2. B_HWTBL : INITIAL(0)	STORES A TRUE OR FALSE VALUE AND HAS NO DEFAULT VALUE. SELECTS WHETHER A SINGLE ARRAY MODULE IS TO BE PM'ED.
3. C_HWTBL : INITIAL(0)	STORES A TRUE OR FALSE VALUE AND HAS NO DEFAULT VALUE. SELECTS WHETHER A SINGLE BANK IS TO BE PM'ED.
4. D_HWTBL : INITIAL(0)	STORES THE SELECTED BANK NUMBER TO BE PM'ED IF PM'ING SINGLE BANKS. THERE IS NO DEFAULT VALUE.
5. E_HWTBL : INITIAL(0)	STORES THE SELECTED ARRAY MODULE TO BE PM'ED IF PM'ING SINGLE ARRAYS. THERE IS NO DEFAULT VALUE.
6. F_HWTBL : INITIAL(176400)	STORES THE RH CONTROLLER BASE REGISTER ADDRESS. THE DEFAULT ADDRESS IS %O'176400'.
7. G_HWTBL : INITIAL(0)	STORES THE DRIVE SELECTION NUMBER OF THE SELECTED DRIVE TO BE PM'ED. THERE IS NO DEFAULT VALUE.
8. H_HWTBL : INITIAL(0)	STORES THE DRIVE OPTION CODE FOR THE SELECTED DRIVE. YES = 16K MOS RAMS NO = 64K MOS RAMS
9. I_HWTBL : INITIAL(TRUE)	STORES A TRUE OF FALSE VALUE. THIS FORCES THE OPERATOR TO REVIEW HIS/HER PARAMETER INPUTS FOR CORRECTNESS

BEFORE PERFORMING PROM MAINTENANCE.

THE DEFAULT VALUE IS TRUE.

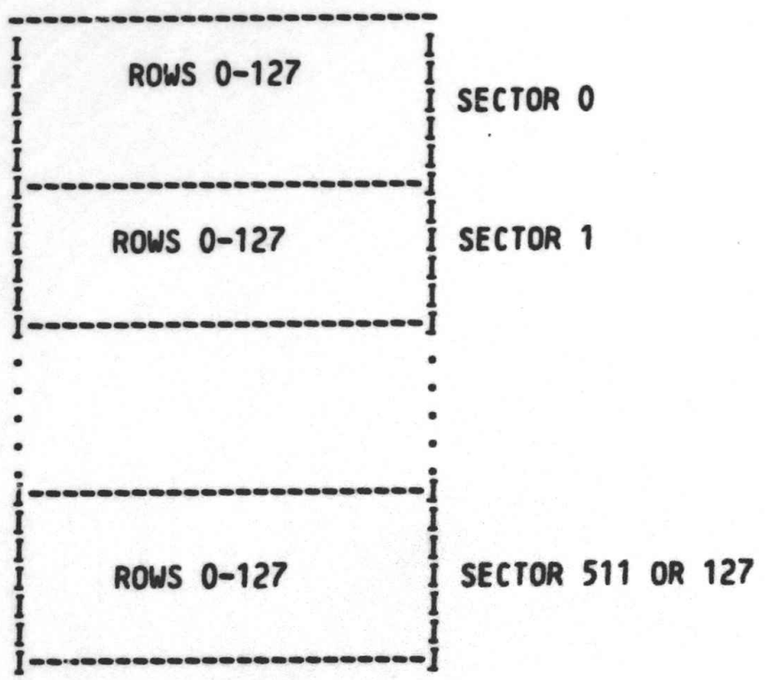
6.0 PROM MAINTENANCE TABLE REPRESENTATIONS

ERROR MAP

THE ERROR MAP IS A BLOCK VECTOR OF 512 BLOCKS.
THESE BLOCKS REPRESENT THE SECTORS IN A CHIP.
EACH BLOCK HAS 8 WORDS. THESE BIT POSITIONS IN
THE WORDS REPRESENT ROW ADDRESSES 0 TO 127.

THE ADJACENT COLUMN ADDRESS FOR EACH ROW ADDRESS
CAN BE CALCULATED BY ADDING THE ROW ADDRESS TO THE
ROWS SECTOR NUMBER.

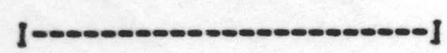
ERROR_MAP

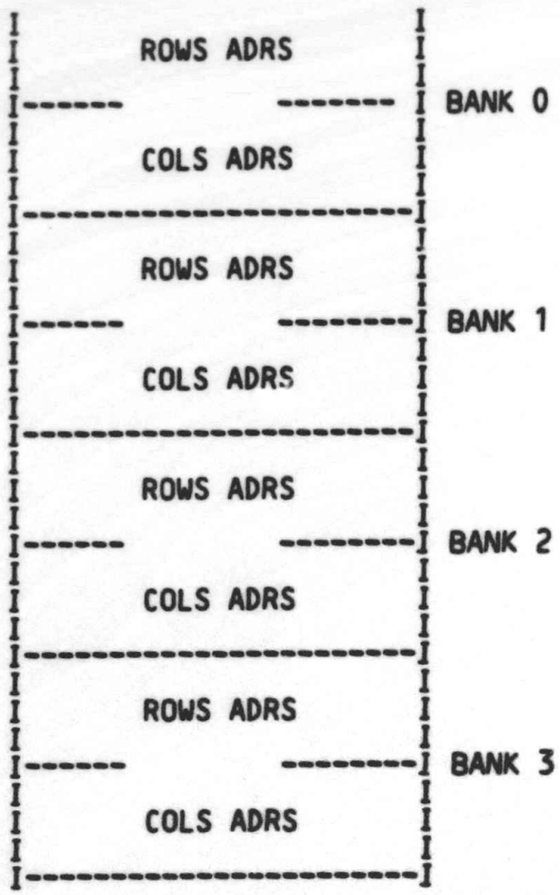


MAIN BLAST TABLE

THE BLAST TABLE IS A BLOCK VECTOR OF FOUR BLOCKS.
EACH BLOCK REPRESENTS PROM BANKS 0 TO 3.
EACH BLOCK HAS 512 WORDS. EACH BLOCK IS FURTHER
DIVIDED INTO TWO SECTIONS. THE UPPER SECTION REPRESENTS
ROW PROM DATA 0 TO 127 (OR 0 TO 256 IF 64K CHIPS)
AND THE LOWER SECTION COLUMN PROM DATA 0 TO 127
(OR 0 TO 256 IF 64K CHIPS).

MAIN BLAST TABLE





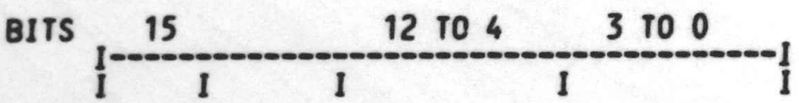
THE BLAST TABLE IS LOADED WITH THE NEW PROM DATA TO MASK OUT THE NEWLY FAILING ROWS AND COLUMNS AND IS ALSO LOADED WITH THE OLD PROM DATA FROM PREVIOUS PROM BLASTING.

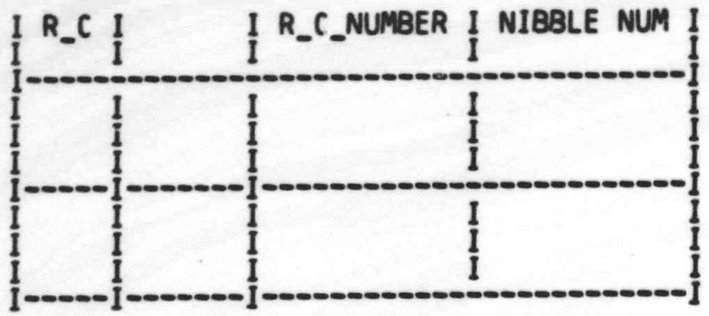
TEMP BLAST TABLE

THE TEMP BLAST TABLE IS A BLOCK OF 10 WORDS. IN EACH WORD IS STORED A BAD ROW OR COLUMNS RESPECTIVE NIBBLE NUMBER, ITS ROW OR COLUMN NUMBER AND A FLAG TO TELL WHETHER THIS TABLE ENTRY IS A ROW ADRS OR A COLUMN ADRS.

THIS TABLE IS NECESSARY BECAUSE ONCE A BAD ROW OR COLUMNS ADDRESS IS STORED INTO THE MAIN BLAST TABLE THERE IS NO WAY OF KNOWING OF WHICH NIBBLE THE ROW OR COLUMN COMES FROM. THIS INFORMATION IS NEEDED WHEN THIS PART IS DETERMINED TO BE ALL BAD (> 10 ALL BAD ROW OR COLUMNS) IN THIS EVENT THIS CHIPS NEWLY FAILING ROWS AND COLUMNS ARE NOT BLASTED. TO ACCOMPLISH THIS THE TEMPORARY BLAST TABLE IS SIMPLY NOT TRANSFERED INTO THE MAIN BLAST TABLE.

TEMP BLAST TABLE





TABLES USED IN CALCULATING SELECTED ROWS AND COLUMNS
FOR BLASTING.

COLUMN COUNT TABLE
THIS TABLE KEEPS A COUNT OF THE NUMBER OF TIMES THE COLUMNS FAILS WITH A PARTICULAR FAILING ROW.

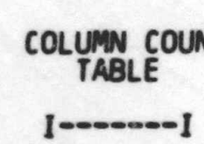
ROW COUNT
THIS SINGLE VARIABLE COUNTS HOW MANY TIMES A ROW NUMBER IS FOUND BAD WHEN SEARCHING THE ERROR MAP. EACH TIME A ROW IS FOUND BAD THE COUNT IS INCREMENTED. IF THIS COUNT GETS > 10 THEN THIS ROW IS CALLED ALL BAD AND IS SELECTED FOR BLASTING. WHEN THIS HAPPENS THE COUNTS OF THE ADJACENT FAILING COLUMN IS DECREMENTED BY ONE AND THIS ROW NUMBER IN THE ERROR MAP IS CLEARED OF ALL OCCURANCES.

IF THE ROW COUNT DOES NOT REACH > 10 AFTER SEARCHING THRU THE ERROR MAP THEN THIS ROW IS NOT SELECTED FOR BLASTING AT THIS TIME AND THE ADJACENT FAILING COLUMN COUNTS ARE NOT DECREMENTED.

COLUMN POINTER TABLE
AS MENTIONED BEFORE THIS TABLE POINTS TO THE ADJACENT FAILING COLUMN NUMBERS WHICH FAILED WITH THE ROW SEARCH.

REMAINDER TABLE
THE PROGRAM EXECUTION WILL FIRST SEARCH THE ERROR MAP FOR ROWS WITH GREATER THAN 10 COUNTS AND THE ADJACENT FAILING COLUMNS COUNTS ARE INCREMENTED. NOW ONCE THE ERROR MAP HAS BEEN SEARCH FOR ROWS COUNTS > 10 THEN COLUMN COUNT TABLE IS SEARCHED FOR COLUMN COUNTS > 10. IF A COLUMN COUNT IS > 10 THEN THIS COLUMN NUMBER IS SELECTED FOR BLASTING AND ITS COUNT IS CLEARED.

ALL THAT IS LEFT NOW ARE RANDOM FAILING ROWS AND COLUMNS SCATTERED THROUGH THE CHIP. THESE SCATTERED ROW COLUMN PAIRS ARE TRANSFERED INTO THE REMAINDER TABLE WHERE THEY CAN BE INTERIGATED AND SELECTED FOR THE BEST POSSIBLE BLASTING CHOICE.



D 2

REMAINDER TABLE

ROW ADRS	COL ADRS

COL 0
COL 1
COL 2
COL 3
COL 4
COL 5
COL 6
COL 7
COL 8
COL 9
COL 10
COL 11
COL XX

ROW COUNT

COLUMN COUNT POINTER TABLE

POINTER 0
POINTER 1
POINTER 2
POINTER 3
POINTER XX

BAD CHIP TABLE

THE BAD CHIP TABLE STORES AWAY FAILING CHIPS DISCOVERED BAD IN A BANK DURING WRITE CHECK TRANSFERS AND DURING SINGLE STEP DMA MODES. THE FAILING DATA PATTERN FOR EACH CHIP IS ALSO STORED.

THIS TABLE IS A BLOCK OF 39 WORDS AND THE BIT DEFINITIONS ARE AS FOLLOWS:

```
BIT 15   IS A FAULT INDICATOR
BIT 14   INDICATES ZEROES DATA FAILURE
BIT 13   INDICATES ONES DATA FAILURE
BIT 12   INDICATES RANDOM DATA 1 FAILURE
BIT 11   INDICATES RANDOM DATA 2 FAILURE
BIT 10   INDICATES RANDOM DATA 3 FAILURE
BIT 9    INDICATES RANDOM DATA 4 FAILURE
BIT 8    INDICATES RANDOM DATA 5 FAILURE
BIT 7    INDICATES RANDOM DATA 6 FAILURE
BIT 6    INDICATES RANDOM DATA 7 FAILURE
BIT 5    INDICATES RANDOM DATA 8 FAILURE
BIT 4    INDICATES RANDOM DATA 9 FAILURE
BIT 3    INDICATES RANDOM DATA 10 FAILURE
BIT 2    INDICATES RANDOM DATA 11 FAILURE
BIT 1    INDICATES RANDOM DATA 12 FAILURE
BIT 0    INDICATES RANDOM DATA 13 FAILURE
```

BAD CHIP TABLE

```
I-------I
I       I CHIP 0
I-------I
I       I CHIP 1
I-------I
I       I CHIP 2
I-------I
I       I CHIP 3
I-------I
I       I CHIP 4
I-------I
I       I CHIP 5
I-------I
I       I CHIP XX
I-------I
I       I CHIP 38
I-------I
```

7.0 TEST SUMMARIES

THIS PROM MAINTENANCE PROGRAM CONTAINS ONLY ONE TEST. THIS ONE TEST IS THE MAIN CONTROL LOOP WHICH LOOPS THE PROGRAM EXECUTION ON THE SELECTED ARRAYS AND BANKS FOR PROM MAINTENANCE.

8.0 MAINTENANCE HISTORY

MODIFIED BY: D.W. NEALE DATE: 18=FEB-82 VERSION: B

MODIFICATIONS TO THIS DIAGNOSTIC ARE PRECEEDED WITH A LINE COMMENT OF '! VERSION CZMLCB'.

```
)%
ELUDOM
```

```
;          0001  MODULE BSKEL2 ( IDENT = 'REV B PATCH 00') =
;          0002  BEGIN
;          0003  REQUIRE 'BLSMAC.REQ';
;          1493  %SBTTL  'PROGRAM HEADER'
;          1494
;          1495
;          1496  LITERAL
;          1497     DS$NBR_OF_TESTS = 1;
;          1498
;          1499
;        C 1500  %(
;        C 1501  ;++
;        C 1502  ; THE PROGRAM HEADER IS THE INTERFACE BETWEEN
;        C 1503  ; THE DIAGNOSTIC PROGRAM AND THE SUPERVISOR.
;        C 1504  ;--
;          1505  )%
;          1506
;          1507  EQUALS;
;          1508
;          1509  POINTER (ALL);
;          1510
;          1511  HEADER  (%ASCII'CZMLCA',%ASCII'B',%ASCII'0',120,0,PRI00);
;          1512
;          1513
;          1514  %SBTTL  'DISPATCH TABLE'
;          1515
;        C 1516  %(
;        C 1517  ;++
;        C 1518  ; THE DISPATCH TABLE CONTAINS THE STARTING ADDRESS OF EACH TEST.
;        C 1519  ; IT IS USED BY THE SUPERVISOR TO DISPATCH TO EACH TEST.
;        C 1520  ;--
;          1521  )%
;          1522
;          1523  DISPATCH (DS$NBR_OF_TESTS);
;          1524
;          1525
;          1526  ERRTBL;
;          1527
;          1528
;          1529
;          1530
;          1531
;          1532
;          1533
;          1534  %SBTTL  'DEFAULT HARDWARE P-TABLE'
;          1535
;        C 1536  %(
;        C 1537  ;++
;        C 1538  ; THE DEFAULT HARDWARE P-TABLE CONTAINS DEFAULT VALUES OF
;        C 1539  ; THE TEST-DEVICE PARAMETERS.  THE STRUCTURE OF THIS TABLE
;        C 1540  ; IS IDENTICAL TO THE STRUCTURE OF THE HARDWARE P-TABLES,
;        C 1541  ; AND IS USED AS A "TEMPLATE" FOR BUILDING THE P-TABLES.
```

```
;       C 1542  ;--
;         1543  )%
;         1544
;         1545  BGNHW   (DFPTBL);
;         1546
;         1547          GLOBAL
;         1548                  A_HWTBL : INITIAL (0),          !MASK ENTIRE SYSTEM FLAG INDICATOR
;         1549                  B_HWTBL : INITIAL (0),          !MASK A SINGLE ARRAY FLAG INDICATOR
;         1550                  C_HWTBL : INITIAL (0),          !MASK A SINGLE BANK FLAG INDICATORG
;         1551                  D_HWTBL : INITIAL (0),          !BANK NO. TO BE MASKED
;         1552                  E_HWTBL : INITIAL (0),          !ARRAY NO. TO BE MASKED
;         1553                  F_HWTBL : INITIAL (%O'176400'), !RH REGISTER BASE ADRS
;         1554                  G_HWTBL : INITIAL (0),          !DRIVE SELECT NO. OF DRIVE TO MASKED
;         1555                  H_HWTBL : INITIAL (1),          !OPTION,  1 = 16K MOS RAMS, 0 = 64K MOS RAM
;         1556                  I_HWTBL: INITIAL (1);           !PARAMETERS CORRECT INDICATOR
;         1557  ENDHW;
;         1558  %SBTTL 'SOFTWARE P-TABLE'
;         1559
;       C 1560  %(
;       C 1561  ;++
;       C 1562  ;  THE SOFTWARE TABLE CONTAINS VARIOUS DATA USED BY THE
;       C 1563  ;  PROGRAM AS OPERATIONAL PARAMETERS.  THESE PARAMETERS ARE
;       C 1564  ;  SET UP A ASSEMBLY TIME AND MAY BE VARIED BY THE OPERATOR
;       C 1565
;       C 1566  ;  AT RUN TIME.  HOWEVER THIS PROGRAM WILL NOT USE THIS OPTIONAL
;       C 1567  ;  SECTION.
;       C 1568  ;--
;         1569  )%
;         1570  !
;         1571  !
;         1572  ! THERE ARE NO SOFTWARE QUESTIONS USED
;         1573  ! DURING THIS PROGRAM.  HOWEVER THIS
;         1574  ! SOFTWARE TABLE LOCATION IS PROVIDED
;         1575  ! IN THE EVENT OF A 'YES' RESPONCE TO
;         1576  ! THE DRS> SOFTWARE QUESTION 'Change Software'
;         1577  !
;         1578  BGNSW (SFPTBL);
;         1579
;         1580  GLOBAL
;         1581      SWTBL$_RET : INITIAL (0);                   !DUMMY ARGUMENT
;         1582
;         1583  ENDSW;
;         1584
;         1585
;         1586
;         1587  %SBTTL  'PROTECTION TABLE'
;         1588
;       C 1589  %(
;       C 1590  ;++
;       C 1591  ; THIS TABLE IS USED BY THE RUNTIME SERVICES
;       C 1592  ; TO PROTECT THE LOAD MEDIA.
;       C 1593  ;--
```

```
;          1594  )%
;          1595
;          1596  BGNPROT(-1,-1,-1);
;          1597
;          1598
;          1599          !1ST ARG =      OFFSET INTO P-TABLE FOR CSR ADDRESS
;          1600          !2ND ARG =      OFFSET INTO P-TABLE FOR MASSBUS ADDRESS
;          1601          !3RD ARG =      OFFSET INTO P-TABLE FOR DRIVE NUMBER
;          1602
;          1603  ENDPROT;
;          1604
;          1605
;          1606  END
;          1607  ELUDOM


                                        .TITLE  BSKEL2
                                        .IDENT  /REV B /


000000                                  .PSECT  $CODE$
000000      103     132     115     L$NAME::.ASCII  /CZM/
000003      114     103     101             .ASCII  /LCA/
000006      000                             .BYTE   0
000007      000                             .BYTE   0
000010                              L$REV::
000010      102                             .ASCII  /B/
000011       060                            .ASCII  /0/
000012  000000G                     L$UNIT::.WORD   T$PTHV
000014  000170                      L$TIML::.WORD   170
000016  000000G                     L$HPCP::.WORD   L$HARD
000020  000000G                     L$SPCP::.WORD   L$SOFT
000022  000140'                     L$HPTP::.WORD   L$HW
000024  000166'                     L$SPTP::.WORD   L$SW
000026  000000G                     L$LADP::.WORD   L$LAST
000030  000000                      L$STA:: .WORD   0
000032  000000                      L$CO::  .WORD   0
000034  000000                      L$DTYP::.WORD   0
000036  000000                      L$APT:: .WORD   0
000040  000124'                     L$DTP:: .WORD   L$DISPATCH
000042  000000                      L$PRIO::.WORD   0
000044  000000                      L$ENVI::.WORD   0
000046  000000                      L$EXP1::.WORD   0
000050                              L$MREV::
000050      003                             .BYTE   3
000051      003                             .BYTE   3
000052  000000                      L$EF::  .WORD   0
000054  000000                              .WORD   0
000056  000000                      L$SPC:: .WORD   0
000060  000000G                     L$DEVP::.WORD   L$DVTYP
000062  000000G                     L$REPP::.WORD   L$RPT
000064  000000                      L$EXP4::.WORD   0
```

```
000066  000000                  L$EXP5::.WORD   0
000070  000000G                 L$AUT:: .WORD   L$AU
000072  000000G                 L$DUT:: .WORD   L$DU
000074  000000                  L$LUN:: .WORD   0
000076  000000G                 L$DESP::.WORD   L$DESC
000100  104035                  L$LOAD::.WORD   -73743
000102  000126'                 L$ETP:: .WORD   L$ERRTBL
000104  000000G                 L$ICP:: .WORD   L$INIT
000106  000000G                 L$CCP:: .WORD   L$CLEAN
000110  000000G                 L$ACP:: .WORD   L$AUTO
000112  000172'                 L$PRT:: .WORD   L$PROT
000114  000000                  L$TEST::.WORD   0
000116  000000                  L$DLY:: .WORD   0
000120  000000                  L$HIME::.WORD   0
000122  000001                  D$PCNT::.WORD   1
000124  000000G                 L$DISPATCH::
                                        .WORD   T1
000126                          ERRTYP::.BLKW   1
000130                          ERRNBR::.BLKW   1
000132                          ERRMSG::.BLKW   1
000134                          ERRBLK::.BLKW   1
000136  000000C                 L$HWLEN::
                                        .WORD   <<L$NDHW-L$HWLEN>/2>
000140  000000                  A.HWTBL::
                                        .WORD   0
000142  000000                  B.HWTBL::
                                        .WORD   0
000144  000000                  C.HWTBL::
                                        .WORD   0
000146  000000                  D.HWTBL::
                                        .WORD   0
000150  000000                  E.HWTBL::
                                        .WORD   0
000152  176400                  F.HWTBL::
                                        .WORD   -1400
000154  000000                  G.HWTBL::
                                        .WORD   0
000156  000001                  H.HWTBL::
                                        .WORD   1
000160  000001                  I.HWTBL::
                                        .WORD   1
000162                          L$NDHW::.BLKW   1
000164  000000C                 L$SWLEN::
                                        .WORD   <<L$NDSW-L$SWLEN>/2>
000166  000000                  SWTBL$.RET::
                                        .WORD   0
000170                          L$NDSW::.BLKW   1
000172  177777                  L$PROT::.WORD   -1
000174  177777                          .WORD   -1
000176  177777                          .WORD   -1
```

```
                        .GLOBL  L$SOFT, T$PTHV, L$RPT, L$INIT
                        .GLOBL  L$CLEAN, L$LAST, L$HARD, L$DVTYP
                        .GLOBL  L$DESC, L$DU, L$AU, L$AUTO, T1


100000                  BIT15==         -100000
040000                  BIT14==         40000
020000                  BIT13==         20000
010000                  BIT12==         10000
004000                  BIT11==         4000
002000                  BIT10==         2000
001000                  BIT09==         1000
000400                  BIT08==         400
000200                  BIT07==         200
000100                  BIT06==         100
000040                  BIT05==         40
000020                  BIT04==         20
000010                  BIT03==         10
000004                  BIT02==         4
000002                  BIT01==         2
000001                  BIT00==         1
001000                  BIT9==          1000
000400                  BIT8==          400
000200                  BIT7==          200
000100                  BIT6==          100
000040                  BIT5==          40
000020                  BIT4==          20
000010                  BIT3==          10
000004                  BIT2==          4
000002                  BIT1==          2
000001                  BIT0==          1
000040                  EF.START==      40
000037                  EF.RESTART==    37
000036                  EF.CONTINUE==   36
000035                  EF.NEW==        35
000034                  EF.PWR==        34
000340                  PRI07==         340
000300                  PRI06==         300
000240                  PRI05==         240
000200                  PRI04==         200
000140                  PRI03==         140
000100                  PRI02==         100
000040                  PRI01==         40
000000                  PRI00==         0
000004                  EVL==           4
000010                  LOT==           10
000020                  ADR==           20
000040                  IDU==           40
000100                  ISR==           100
000200                  UAM==           200
000400                  BOE==           400
001000                  PNT==           1000
```

```
        002000                  PRI==           2000
        004000                  IXE==           4000
        010000                  IBE==           10000
        020000                  IER==           20000
        040000                  LOE==           40000
        100000                  HOE==           -100000
        000126'                 L$ERRTBL==      ERRTYP
        000166'                 L$SW==          L$SWLEN+2
        000140'                 L$HW==          L$HWLEN+2
        000011'                 L$DEPO==        L$REV+1
        000140'                 DFPTBL==        L$HWLEN+2
        000166'                 SFPTBL==        L$SWLEN+2



; Size:          0 code + 64 data words
; Run Time:      00:03.4
; Elapsed Time: 00:10.0
; Memory Used:  29 pages
; Compilation Complete
```

```
;          0001  MODULE BSKEL3 (IDENT = 'REV B PATCH 00') =
;          0002  BEGIN
;          0003  REQUIRE 'BLSMAC.REQ';
;          1493  EQUALS;
;          1494  EXTERNAL ROUTINE
;          1495      SUMMARY : NOVALUE;                    !SUMMARY REPORT CODE ROUTINE
;          1496
;          1497  %SBTTL 'TYPE AND DESCRIPTION'
;          1498
;          1499
;          1500
;          1501
;          1502  DEVTYP  (%ASCIZ'ML-11 BLOCK MODE MEMORY SYSTEM');
;          1503
;          1504
;          1505  DESCRIPT (%ASCIZ'ML-11 PROM MAINTENANCE PROGRAM');
;          1506
;          1507
;          1508
;          1509  %SBTTL 'HARDWARE PARAMETER CODING SECTION'
;          1510
;        C 1511  %(
;        C 1512  ;++
;        C 1513  ; THE HARDWARE PARAMETER CODING SECTION CONTAINS MACROS
;        C 1514  ; THAT ARE USED BY THE SUPERVISOR TO BUILD P-TABLES.  THE
;        C 1515  ; MACROS ARE NOT EXECUTED AS MACHINE INSTRUCTIONS BUT ARE
;        C 1516  ; INTERPRETED BY THE SUPERVISOR AS DATA STRUCTURES.  THE
;        C 1517  ; MACROS ALLOW THE SUPERVISOR TO ESTABLISH COMMUNICATIONS
;        C 1518  ; WITH THE OPERATOR.
;        C 1519  ;--
;          1520  )%
;          1521
;          1522  BGNHRD;
;          1523      BIND
;          1524          !
;          1525          ! DEFINE HARDWARE MESSAGES
;          1526          !
;          1527          M_SYS = UPLIT(%ASCIZ'IS ENTIRE ML-11 SYSTEM TO BE MASKED ?'),
;          1528          M_ARR = UPLIT(%ASCIZ'IS A SINGLE ARRAY TO BE MASKED      ?'),
;          1529          M_BNK = UPLIT(%ASCIZ'IS A SINGLE BANK TO BE MASKED       ?'),
;          1530          M_BNK_NO = UPLIT(%ASCIZ'ENTER BANK NO. TO BE MASKED        ?'),
;          1531          M_BRD_NO = UPLIT(%ASCIZ'ENTER BOARD NO. TO BE MASKED       ?'),
;          1532          M_RH_BASE = UPLIT(%ASCIZ'STARTING RH BASE REGISTER ADDRESS  ?'),
;          1533          M_DUT = UPLIT(%ASCIZ'DUT DRIVE NUMBER                    ?'),
;          1534          M_OPTION = UPLIT(%ASCIZ'IS DRIVE OPTION AN ML11A         ?'),
;          1535          M_CORRECT = UPLIT(%ASCIZ'ARE YOUR INPUTED PARAMETERS CORRECT ?');
;          1536          !
;          1537          ! SELECT WHICH OPTION THE PROGRAM IS
;          1538          ! TO RUN UNDER.
;          1539          !       OPTION 1. PROM MAINT ENTIRE ML11 SYSTEM
;          1540          !       OPTION 2. PROM MAINT A SINGLE ARRAY
;          1541          !       OPTION 3. PROM MAINT A SINGLE BANK
```

```
;          1542          !
;          1543          ! ASK AM I TO MASK AN ENTIRE SYSTEM ?
;          1544          !
;          1545          GPRML(M_SYS,%O'0',1,NO,1);
;          1546          XFERT(DRV_PRAM);
;          1547          !
;          1548          ! ASK AM I TO MASK AN ENTIRE ARRAY ?
;          1549          !
;          1550          GPRML(M_ARR,%O'2',1,NO,1);
;          1551          XFERT(ENTER_BOARD);
;          1552          !
;          1553          ! ASK AM I TO MASK A SINGLE BANK ?
;          1554          !
;          1555          GPRML(M_BNK,%O'4',1,NO,1);
;          1556          !
;          1557          ! INPUT TO THE PROGRAM THE SELECTED
;          1558          ! PROGRAM RUN TIME PARAMETERS.
;          1559          !
;          1560          XFERF(DONE_HRD);
;          1561          !
;          1562          ! ENTIRE BANK TO BE PROM MAINT
;          1563          !
;          1564          GPRMD(M_BNK_NO,%O'6',D,%O'7',0,%DECIMAL'3',NO,1);
;          1565      $L(ENTER_BOARD);
;          1566          !
;          1567          ! ENTIRE BOARD TO BE PROM MAINT
;          1568          !
;          1569          GPRMD(M_BRD_NO,%O'10',D,%O'77',0,%DECIMAL'15',NO,1);
;          1570      $L(DRV_PRAM);
;          1571          !
;          1572          ! ENTIRE RH BASE ADDRESS
;          1573          ! ENTIRE DRIVE UNDER TEST
;          1574          ! ENTIRE DRIVE OPTION 16K PARTS OR 64K PARTS
;          1575          !
;          1576          GPRMA(M_RH_BASE,%O'12',0,0,%O'177777',YES,1);
;          1577          GPRMD(M_DUT,%O'14',0,%O'7',0,%O'77',NO,1);
;          1578          GPRML(M_OPTION,%O'16',1,NO,1);
;          1579          !
;          1580          ! FORCE THE OPERATOR TO REVIEW HIS INPUTED
;          1581          ! PARAMETERS FOR CORRECTNESS BY ASKING THE
;          1582          ! NEXT QUESTION.  THE INIT CODE WILL ABORT
;          1583          ! THE PROGRAM EXECUTION IF HE RETURNS A NO
;          1584          ! RESPONCE
;          1585          !
;          1586      $L(DONE_HRD);
;          1587          GPRML(M_CORRECT,%O'20',1,YES,1);
;          1588  ENDHRD;
;          1589
;          1590  %SBTTL 'SOFTWARE PARAMETER CODING SECTION'
;          1591
;        C 1592  %(
;        C 1593  ;++
```

```
;       C 1594  ;  THE SOFTWARE PARAMETER CODING SECTION CONTAINS MACROS
;       C 1595  ;  THAT ARE USED BY THE SUPERVISOR TO BUILD P-TABLES.  THE
;       C 1596  ;  MACROS ARE NOT EXECUTED AS MACHINE INSTRUCTIONS BUT ARE
;       C 1597  ;  INTERPRETED BY THE SUPERVISOR AS DATA STRUCTURES.  THE
;       C 1598  ;  MACROS ALLOW THE SUPERVISOR TO ESTABLISH COMMUNICATIONS
;       C 1599  ;  WITH THE OPERATOR.  HOWEVER THIS PROGRAM WILL NOT USE THIS SECTION.
;       C 1600  ;--
;         1601  )%

;         1602
;         1603  BGNSFT;
;         1604      BIND
;         1605
;         1606              ! DEFINE THE SOFTWARE MESSAGES
;         1607              !
;         1608
;         1609              !
;         1610              M_RET = UPLIT(%ASCIZ' NOT USED TYPE <CR>');
;         1611
;         1612              ! INPUT THE SOFTWARE QUESTION TO THE PROGRAM
;         1613              !
;         1614              GPRML(M_RET,0,1,YES,1);
;         1615
;         1616  ENDSFT;
;         1617
;         1618  %SBTTL 'REPORT CODING SECTION'
;         1619
;         1620
;         1621
;       C 1622  %(
;       C 1623  !+
;       C 1624  ! THE REPORT CODING SECTION CONTAINS THE
;       C 1625  ! 'PRINTS' CALLS THAT GENERATE STATISTICAL REPORTS.
;       C 1626  !-
;         1627  )%
;         1628  BGNRPT;
;         1629  !
;         1630  ! THE SUMMARY REPORT CODE ROUTINE FOR THIS
;         1631  ! PROGRAM IS LOCATED IN PMSKL4 OF THIS
;         1632  ! DIAGNOSTIC.
;         1633  !
;         1634
;         1635  SUMMARY ();                    !CALL THE REPORT SUMMARY ROUTINE
;         1636  RETURN;
;         1637
;         1638  ENDRPT;


                                               .TITLE  BSKEL3
                                               .IDENT  /REV B /

000000                                         .PSECT  $CODE$
```

```
000000      115      114      055          L$DVTYP::
                                                .ASCII  /ML-/
000003      061      061      040                .ASCII  /11 /
000006      102      114      117                .ASCII  /BLO/
000011      103      113      040                .ASCII  /CK /
000014      115      117      104                .ASCII  /MOD/
000017      105      040      115                .ASCII  /E M/
000022      105      115      117                .ASCII  /EMO/
000025      122      131      040                .ASCII  /RY /
000030      123      131      123                .ASCII  /SYS/
000033      124      105      115                .ASCII  /TEM/
000036      000      000                         .ASCII  <00><00>
000040      115      114      055          L$DESC::.ASCII  /ML-/
000043      061      061      040                .ASCII  /11 /
000046      120      122      117                .ASCII  /PRO/
000051      115      040      115                .ASCII  /M M/
000054      101      111      116                .ASCII  /AIN/
000057      124      105      116                .ASCII  /TEN/
000062      101      116      103                .ASCII  /ANC/
000065      105      040      120                .ASCII  /E P/
000070      122      117      107                .ASCII  /ROG/
000073      122      101      115                .ASCII  /RAM/
000076      000      000                         .ASCII  <00><00>
000100  000000C                          L$HRDLN::
                                                .WORD   <<<L$NDHRD-L$HRDLN>/2>-1>
000102  000120                           GP$1::  .WORD   120
000104  000000'                                 .WORD   M.SYS
000106  000001                                  .WORD   1
000110  000000C                          $DRV.PRAM:
                                                .WORD   <<<<$LDRV.PRAM-$DRV.PRAM>*400>+4>+20>
000112  001120                           GP$2::  .WORD   1120
000114  000046'                                 .WORD   M.ARR
000116  000001                                  .WORD   1
000120  000000C                          $ENTER.BOARD:
                                                .WORD   <<<<$LENTER.BOARD-$ENTER.BOARD>*400>+4>+20>
000122  002120                           GP$3::  .WORD   2120
000124  000114'                                 .WORD   M.BNK
000126  000001                                  .WORD   1
000130  000000C                          $DONE.HRD:
                                                .WORD   <<<<$LDONE.HRD-$DONE.HRD>*400>+4>+40>
000132  003042                           GP$4::  .WORD   3042
000134  000162'                                 .WORD   M.BNK.NO
000136  000007                                  .WORD   7
000140  000000                                  .WORD   0
000142  000003                                  .WORD   3
000144  001004                           $LENTER.BOARD:
                                                .WORD   1004
000146  004042                           GP$5::  .WORD   4042
000150  000230'                                 .WORD   M.BRD.NO
000152  000077                                  .WORD   77
000154  000000                                  .WORD   0
000156  000017                                  .WORD   17
```

```
000160  001004                          $LDRV.PRAM:
                                                .WORD   1004
000162  005031                         GP$6::  .WORD   5031
000164  000276'                                 .WORD   M.RH.BASE
000166  000000                                  .WORD   0
000170  177777                                  .WORD   -1
000172  006022                         GP$7::  .WORD   6022
000174  000344'                                 .WORD   M.DUT
000176  000007                                  .WORD   7
000200  000000                                  .WORD   0
000202  000077                                  .WORD   77
000204  007120                         GP$8::  .WORD   7120
000206  000376'                                 .WORD   M.OPTION
000210  000001                                  .WORD   1
000212  001004                          $LDONE.HRD:
                                                .WORD   1004
000214  010130                         GP$9::  .WORD   10130
000216  000436'                                 .WORD   M.CORRECT
000220  000001                                  .WORD   1
000222                                  L$NDHRD::
                                                .BLKW   1
000224  000000C                         L$SFTLN::
                                                .WORD   <<<L$NDSFT-L$SFTLN>/2>-1>
000226  000130                         GP$10:: .WORD   130
000230  000504'                                 .WORD   M.RET
000232  000001                                  .WORD   1
000234                                  L$NDSFT::
                                                .BLKW   1



000000                                          .PSECT  $PLIT$,  D
000000  111     123     040             P.AAA:  .ASCII  /IS /
000003  105     116     124                     .ASCII  /ENT/
000006  111     122     105                     .ASCII  /IRE/
000011  040     115     114                     .ASCII  / ML/
000014  055     061     061                     .ASCII  /-11/
000017  040     123     131                     .ASCII  / SY/
000022  123     124     105                     .ASCII  /STE/
000025  115     040     124                     .ASCII  /M T/
000030  117     040     102                     .ASCII  /O B/
000033  105     040     115                     .ASCII  /E M/
000036  101     123     113                     .ASCII  /ASK/
000041  105     104     040                     .ASCII  /ED /
000044  077     000                             .ASCII  /?/<00>
000046  111     123     040             P.AAB:  .ASCII  /IS /
000051  101     040     123                     .ASCII  /A S/
000054  111     116     107                     .ASCII  /ING/
000057  114     105     040                     .ASCII  /LE /
000062  101     122     122                     .ASCII  /ARR/
000065  101     131     040                     .ASCII  /AY /
000070  124     117     040                     .ASCII  /TO /
```

```
000073  102  105  040                .ASCII  /BE /
000076  115  101  123                .ASCII  /MAS/
000101  113  105  104                .ASCII  /KED/
000104  040  040  040                .ASCII  /   /
000107  040  040  040                .ASCII  /   /
000112  077  000                     .ASCII  /?/<00>
000114  111  123  040       P.AAC:   .ASCII  /IS /
000117  101  040  123                .ASCII  /A S/
000122  111  116  107                .ASCII  /ING/
000125  114  105  040                .ASCII  /LE /
000130  102  101  116                .ASCII  /BAN/
000133  113  040  124                .ASCII  /K T/
000136  117  040  102                .ASCII  /O B/
000141  105  040  115                .ASCII  /E M/
000144  101  123  113                .ASCII  /ASK/
000147  105  104  040                .ASCII  /ED /
000152  040  040  040                .ASCII  /   /
000155  040  040  040                .ASCII  /   /
000160  077  000                     .ASCII  /?/<00>
000162  105  116  124       P.AAD:   .ASCII  /ENT/
000165  105  122  040                .ASCII  /ER /
000170  102  101  116                .ASCII  /BAN/
000173  113  040  116                .ASCII  /K N/
000176  117  056  040                .ASCII  /O. /
000201  124  117  040                .ASCII  /TO /
000204  102  105  040                .ASCII  /BE /
000207  115  101  123                .ASCII  /MAS/
000212  113  105  104                .ASCII  /KED/
000215  040  040  040                .ASCII  /   /
000220  040  040  040                .ASCII  /   /
000223  040  040  040                .ASCII  /   /
000226  077  000                     .ASCII  /?/<00>
000230  105  116  124       P.AAE:   .ASCII  /ENT/
000233  105  122  040                .ASCII  /ER /
000236  102  117  101                .ASCII  /BOA/
000241  122  104  040                .ASCII  /RD /
000244  116  117  056                .ASCII  /NO./
000247  040  124  117                .ASCII  / TO/
000252  040  102  105                .ASCII  / BE/
000255  040  115  101                .ASCII  / MA/
000260  123  113  105                .ASCII  /SKE/
000263  104  040  040                .ASCII  /D  /
000266  040  040  040                .ASCII  /   /
000271  040  040  040                .ASCII  /   /
000274  077  000                     .ASCII  /?/<00>
000276  123  124  101       P.AAF:   .ASCII  /STA/
000301  122  124  111                .ASCII  /RTI/
000304  116  107  040                .ASCII  /NG /
000307  122  110  040                .ASCII  /RH /
000312  102  101  123                .ASCII  /BAS/
000315  105  040  122                .ASCII  /E R/
000320  105  107  111                .ASCII  /EGI/
```

```
000323  123  124  105                    .ASCII  /STE/
000326  122  040  101                    .ASCII  /R A/
000331  104  104  122                    .ASCII  /DDR/
000334  105  123  123                    .ASCII  /ESS/
000337  040  040  040                    .ASCII  /   /
000342  077  000                        .ASCII  /?/<00>
000344  104  125  124          P.AAG:    .ASCII  /DUT/
000347  040  104  122                    .ASCII  / DR/
000352  111  126  105                    .ASCII  /IVE/
000355  040  116  125                    .ASCII  / NU/
000360  115  102  105                    .ASCII  /MBE/
000363  122  011  011                    .ASCII  /R/<11><11>
000366  040  040  040                    .ASCII  /   /
000371  040  040  077                    .ASCII  /  ?/
000374  000  000                        .ASCII  <00><00>
000376  111  123  040          P.AAH:    .ASCII  /IS /
000401  104  122  111                    .ASCII  /DRI/
000404  126  105  040                    .ASCII  /VE /
000407  117  120  124                    .ASCII  /OPT/
000412  111  117  116                    .ASCII  /ION/
000415  040  101  116                    .ASCII  / AN/
000420  040  115  114                    .ASCII  / ML/
000423  061  061  101                    .ASCII  /11A/
000426  011  040  040                    .ASCII  <11>/  /
000431  040  040  040                    .ASCII  /   /
000434  077  000                        .ASCII  /?/<00>
000436  101  122  105          P.AAI:    .ASCII  /ARE/
000441  040  131  117                    .ASCII  / YO/
000444  125  122  040                    .ASCII  /UR /
000447  111  116  120                    .ASCII  /INP/
000452  125  124  105                    .ASCII  /UTE/
000455  104  040  120                    .ASCII  /D P/
000460  101  122  101                    .ASCII  /ARA/
000463  115  105  124                    .ASCII  /MET/
000466  105  122  123                    .ASCII  /ERS/
000471  040  103  117                    .ASCII  / CO/
000474  122  122  105                    .ASCII  /RRE/
000477  103  124  040                    .ASCII  /CT /
000502  077  000                        .ASCII  /?/<00>
000504  040  116  117          P.AAJ:    .ASCII  / NO/
000507  124  040  125                    .ASCII  /T U/
000512  123  105  104                    .ASCII  /SED/
000515  040  124  131                    .ASCII  / TY/
000520  120  105  040                    .ASCII  /PE /
000523  074  103  122                    .ASCII  /<CR/
000526  076  000                        .ASCII  />/<00>

                                        .GLOBL  SUMMARY

100000                          BIT15==             -100000
```

```
040000          BIT14==         40000
020000          BIT13==         20000
010000          BIT12==         10000
004000          BIT11==         4000
002000          BIT10==         2000
001000          BIT09==         1000
000400          BIT08==         400
000200          BIT07==         200
000100          BIT06==         100
000040          BIT05==         40
000020          BIT04==         20
000010          BIT03==         10
000004          BIT02==         4
000002          BIT01==         2
000001          BIT00==         1
001000          BIT9==          1000
000400          BIT8==          400
000200          BIT7==          200
000100          BIT6==          100
000040          BIT5==          40
000020          BIT4==          20
000010          BIT3==          10
000004          BIT2==          4
000002          BIT1==          2
000001          BIT0==          1
000040          EF.START==      40
000037          EF.RESTART==    37
000036          EF.CONTINUE==   36
000035          EF.NEW==        35
000034          EF.PWR==        34
000340          PRI07==         340
000300          PRI06==         300
000240          PRI05==         240
000200          PRI04==         200
000140          PRI03==         140
000100          PRI02==         100
000040          PRI01==         40
000000          PRI00==         0
000004          EVL==           4
000010          LOT==           10
000020          ADR==           20
000040          IDU==           40
000100          ISR==           100
000200          UAM==           200
000400          BOE==           400
001000          PNT==           1000
002000          PRI==           2000
004000          IXE==           4000
010000          IBE==           10000
020000          IER==           20000
040000          LOE==           40000
100000          HOE==           -100000
```

```
        000102'                         L$HARD==        L$HRDLN+2
        000000'                         M.SYS=          P.AAA
        000046'                         M.ARR=          P.AAB
        000114'                         M.BNK=          P.AAC
        000162'                         M.BNK.NO=       P.AAD
        000230'                         M.BRD.NO=       P.AAE
        000276'                         M.RH.BASE=      P.AAF
        000344'                         M.DUT=          P.AAG
        000376'                         M.OPTION=       P.AAH
        000436'                         M.CORRECT=      P.AAI
        000226'                         L$SOFT==        L$SFTLN+2
        000504'                         M.RET=          P.AAJ


                                        .SBTTL  LRPT REPORT CODING SECTION
000236                                  .PSECT  $CODE$

000236  004767  000000G       LRPT:     JSR     PC,SUMMARY                ;                1635
000242  000207                          RTS     PC                        ;                1616

; Routine Size:  3 words
; Maximum stack depth per invocation:  0 words



                                        .SBTTL  L$RPT REPORT CODING SECTION
000244  004767  177766        L$RPT::   JSR     PC,LRPT                   ;                1636
000250  104425                          TRAP    25
000252  000207                          RTS     PC

; Routine Size:  4 words
; Maximum stack depth per invocation:  0 words


;           1639
;           1640  %SBTTL  'AUTODROP SECTION'
;           1641
;         C 1642  %(
;         C 1643  !+
;         C 1644  ! THIS CODE IS EXECUTED IMMEDIATELY AFTER THE INITIALIZE CODE IF
;         C 1645  ! THE "ADR" FLAG WAS SET.  THE UNIT(S) UNDER TEST ARE CHECKED TO
;         C 1646  ! SEE IF THEY WILL RESPOND.  THOSE THAT DON'T ARE IMMEDIATELY
;         C 1647  ! DROPPED FROM TESTING.
;         C 1648  !-
;           1649  )%
;           1650
;           1651  ! THIS SECTION 'AUTO DROP' IS NOT USED
;           1652  ! DURING THIS PROGRAM
;           1653  !
;           1654  BGNAUTO;
;           1655  RETURN;
```

```
;         1656  ENDAUTO;


                                        .SBTTL  LAUTO AUTODROP SECTION
000254  000207                  LAUTO:  RTS     PC                      ;                                       1638

; Routine Size:  1 word
; Maximum stack depth per invocation:  0 words



                                        .SBTTL  L$AUTO AUTODROP SECTION
000256  004767  177772          L$AUTO::JSR     PC,LAUTO                ;                                       1655
000262  104461                         TRAP    61
000264  000207                         RTS     PC

; Routine Size:  4 words
; Maximum stack depth per invocation:  0 words


;         1657
;         1658
;         1659  %SBTTL  'DROP UNIT SECTION'
;       C 1661  %(
;       C 1662  !+
;       C 1663  ! THE DROP-UNIT SECTION CONTAINS THE CODING THAT CAUSES A DEVICE
;       C 1664  ! TO NO LONGER BE TESTED.
;       C 1665  !-
;         1666  )%
;         1667  ! THIS SECTION 'DROP UNIT' IS NOT USED
;         1668  ! DURING THIS PROGRAM
;         1669  !
;         1670  BGNDU;
;         1671  RETURN;
;         1672  ENDDU;


                                        .SBTTL  LDU DROP UNIT SECTION
000266  000207                  LDU:    RTS     PC                      ;                                       1656

; Routine Size:  1 word
; Maximum stack depth per invocation:  0 words



                                        .SBTTL  L$DU DROP UNIT SECTION
000270  004767  177772          L$DU::  JSR     PC,LDU                  ;                                       1671
000274  104453                         TRAP    53
000276  000207                         RTS     PC
```

```
; Routine Size:  4 words
; Maximum stack depth per invocation:  0 words


;          1673
;          1674
;          1675  %SBTTL  'ADD UNIT SECTION'
;          1676
;        C 1677  %(
;        C 1678  !+
;        C 1679  ! THE ADD-UNIT SECTION CONTAINS ANY CODE THE PROGRAMMER WISHES
;        C 1680  !  TO BE EXECUTED IN CONJUNCTION WITH THE ADDING OF A UNIT BACK
;        C 1681  !  TO THE TEST CYCLE.
;        C 1682  !-
;          1683  )%
;          1684  ! THIS SECTION 'ADD UNIT' IS NOT USED
;          1685  ! DURING THIS PROGRAM
;          1686  !
;          1687  BGNAU;
;          1688  RETURN;
;          1689  ENDAU;

                                        .SBTTL  LAU ADD UNIT SECTION
000300  000207                  LAU:    RTS     PC                      ;  1672

; Routine Size:  1 word
; Maximum stack depth per invocation:  0 words



                                        .SBTTL  L$AU ADD UNIT SECTION
000302  004767  177772          L$AU::  JSR     PC,LAU                  ;  1688
000306  104452                          TRAP    52
000310  000207                          RTS     PC

; Routine Size:  4 words
; Maximum stack depth per invocation:  0 words


;          1690
;          1691
;          1692
;          1693  END
;          1694  ELUDOM
```

```
; Size:          22 code + 251 data words
; Run Time:      00:04.6
; Elapsed Time: 00:08.3
; Memory Used:   30 pages
; Compilation Complete
```

```
                0001  MODULE BSKEL4 (IDENT = 'REV B PATCH 00'
                0002                ) =
                0003  BEGIN
                0004  !
                0005  ! PRETTY BLF COMMANDS
                0006  !
                0007  !<BLF/LOWERCASE_KEY>
                0008  !
                0009  ! LIBRARY FILES
                0010  !
                0011  !
                0012  ! REQUIRE FILES
                0013  !
                0014
                0015  require 'BLSMAC.REQ';
                1505
              C 1506  %(
              C 1507  DUE TO THE BLISS-16 COMPILER BLISS16 COMPILERS  RESTRICTION OF 6 CHARACTERS PER UNIQUE VARIABLE NAME
              C 1508  THE FOLLOWING ROUTINE SUBSCRIPTS ARE USED:
              C 1509          !
              C 1510          ! ROUTINE SUBSCPIPTS
              C 1511          !
              C 1512          I_ROUTINE NAME = INITIALIZE
              C 1513          IN_ROUTINE NAME = INTERIGATE
              C 1514          L_ROUTINE NAME = LOAD
              C 1515          DM_ROUTINE NAME = DIAGNOSTIC MODE
              C 1516          GEN_ROUTINE NAME = GENERATE
              C 1517          FB_ROUTINE NAME = FIND BAD
              C 1518          CLR_ROUTINE NAME = CLEAR
              C 1519          OFF_ROUTINE NAME = OFFSETS
              C 1520          MOV_ROUTINE NAME = MOVE
              C 1521          CS_ROUTINE NAME = CHECK SUM
              C 1522          PM_ROUTINE NAME = PROM MAINTANENCE
              C 1523          CAL_ROUTINE NAME = CALCULATE
              C 1524          BL_ROUTINE NAME = BLAST
              C 1525          VER_ROUTINE NAME = VERIFY
              C 1526          S_ROUTINE NAME = SEARCH
              C 1527          X_ROUTINE NAME = TRANSFER
              C 1528          !
              C 1529          ! MISCELLANEOUS SUBSCRIPTS
              C 1530          !
              C 1531          F_MISCELANEOUS NAME = FLAG
                1532  )%
```

```
;          1533  %sbttl 'CONSTANT LITERAL DECLARATIONS'
;          1534  !
;          1535  ! CONSTANT LITERAL DECLARATIONS
;          1536  !
;          1537
;          1538  literal
;          1539      !
;          1540      ! Mass bus transfer codes
;          1541      !
;          1542      func_1 = %O'000001',                !Noop function
;          1543      FUNC_2 = %o'000011',                !Drive clear function
;          1544      FUNC_3 = %o'000031',                !Search function
;          1545      FUNC_4 = %o'000051',                !Write check function
;          1546      FUNC_5 = %o'000061',                !Write function
;          1547      FUNC_6 = %o'000071',                !Read function
;          1548      !
;          1549      ! DATA CONSTANTS
;          1550      !
;          1551      ONES = %o'177777',                  !All ones data field
;          1552      ONE = %o'000001',                   !Single one data field
;          1553      ZEROES = %o'000000',                !All zeroes data field
;          1554      ZERO = %o'000000',                  !Zero data field
;          1555      !
;          1556      ! REGISTER ACCESS OFFSET INDEXES
;          1557      !
;          1558      MLCS1 = 0,                          !ML_ADDR + %O'0' CONTROL AND STATUS REGISTER 1
;          1559      MLWC = 1,                           !ML_ADDR + %O'2' WORD COUNT REGISTER
;          1560      MLBA = 2,                           !ML_ADDR + %O'4' UNIBUS ADDRESS REGISTER
;          1561      MLDA = 3,                           !ML_ADDR + %O'6' DESIRED ADDRESS REGISTER
;          1562      MLCS2 = 4,                          !ML_ADDR + %O'10' CONTROL AND STATUS REGISTER 2
;          1563      MLDS = 5,                           !ML_ADDR + %O'12' DRIVE STATUS REGISTER
;          1564      MLER = 6,                           !ML_ADDR + %O'14' ERROR REGISTER
;          1565      MLAS = 7,                           !ML_ADDR + %O'16' ATTENTION SUMMARY REGISTER
;          1566      MLLA = 8,                           !ML_ADDR + %O'20' LOOK AHEAD REGISTER
;          1567      MLPA = 8,                           !ML_ADDR + %O'20' PROM ADDRESS REGISTER
;          1568      MLDB = 9,                           !ML_ADDR + %O'22' DATA BUFFER REGISTER
;          1569      MLMR = 10,                          !ML_ADDR + %O'24' MAINTENANCE REGISTER
;          1570      MLDT = 11,                          !ML_ADDR + %O'26' DRIVE TYPE REGISTER
;          1571      MLSN = 12,                          !ML_ADDR + %O'30' SERIAL NUMBER REGISTER
;          1572      MLE1 = 13,                          !ML_ADDR + %O'32' ECC CRC WORD REGISTER 1
;          1573      MLE2 = 14,                          !ML_ADDR + %O'34' ECC CRC WORD REGISTER 2
;          1574      MLD1 = 15,                          !ML_ADDR + %O'36' DATA DIAGNOSTIC REGISTER 1
;          1575      MLD2 = 16,                          !ML_ADDR + %O'40' DATA DIAGNOSTIC REGISTER 2
;          1576      MLEE = 17,                          !ML_ADDR + %O'42' ECC ERROR REGISTER
;          1577      MLEL = 18,                          !ML_ADDR + %O'44' ECC ERROR LOCATION REGISTER
;          1578      MLPD = 19,                          !ML_ADDR + %O'46' PROM DATA REGISTER
;          1579      MLBAE = 20,                         !ML_ADDR + %O'50' BUS ADDRESS E52TENSION REGISTER
;          1580      MLCS3 = 21,                         !ML_ADDR + %O'52' CONTROL AND STATUS REGISTER 3
;          1581      !
;          1582      ! ERROR NUMBER CONSTANTS
;          1583      !
;          1584
;                                                        ! **** ERROR LOCATION ****
```

```
1585        ERR_1 = 1,                              !Init code section
1586        ERR_2 = 2,                              !Init code section
1587        ERR_3 = 3,                              !Init code section
1588        ERR_4 = 4,                              !Pm_this_bank routine
1589        ERR_5 = 5,                              !In_error_map routine
1590        ERR_6 = 6,                              !In_error_map routine
1591        ERR_7 = 7,                              !In_blast_tbl routine
1592        ERR_8 = 8,                              !Ver_blast routine
1593        ERR_9 = 9,                              !Init code section
1594        ERR_10 = 10,                            !Bl_proms routine
1595        ERR_11 = 11,                            !Ver_error_mask routine
1596        ERR_12 = 12,                            !Ver_error_mask routine
1597        ERR_13 = 13,                            !Mass bus transfer routine
1598        !
1599        ! DESIRED SECTOR ADDRESS FIELD SELECT CONSTANT VALUES
1600        !
1601        bnk$sel_size = 2,                       !Bank select size expression
1602        ARR$SEL_SIZE = 4,                       !Array select size expression
1603        !
1604        ! FLAG REGISTER SELECTION DEFINITION
1605        !
1606        f_flg_err = 0,                          !General purpose error flg
1607        F_UNC_ERR_FLG = 1,                      !Indicates an uncorrectable error was detected
1608        F_ERR_MAP_ENTERED = 2,                  !Indicates that the error map has an entry
1609        F_BLST_TBL_ENTERED = 3,                 !Indicates that the blast table was entered
1610        F_ALL_BAD_CHIP = 4,             !Indicates that this bank has one all bad chip >14 row or col bad
1611        F_RAND_DATA = 5,                        !Indicates that random data pattern is presently used
1612        F_ABORT_ARRAY = 6,              !Indicates that further testing of this array is to be aborted
1613        F_D_CLK_TIME_OUT = 7,                   !Indicates that the data clock bit is hung high
1614        !
1615        ! BOOLEAN VALUES
1616        !
1617        TRUE = 1,                               !Logical true indicator
1618        FALSE = 0,                              !Logical false indicator
1619        !
1620        ! CRC DATA BIT DEFINITIONS
1621        !
1622        CRC_P = 36,                             !Ecc crc chip 36
1623        CRC_A = 37,                             !Ecc crc chip 37
1624        CRC_B = 38,                             !Ecc crc chip 38
1625        CRC_NIBBLE = 9,                         !Ecc crc nibble nine
1626        !
1627        ! DELAY MACRO DELAY VARIABLES
1628        !
1629        one_us = 1,                             !delay for one micro second
1630        FIFTY_MS = 100,                         !Delay for 50 milli seconds
1631        TEN_MS = 10000,                         !Delay for 10 milli seconds
1632        !
1633        ! MISCELLANIOUS CONSTANTS
1634        !
1635        ml11a = 1,                              !ML11A is a 16k mos ram array
1636        SET_FLG = 1,                            !Constant to set a flag
```

```
;         1637      CLR_FLG = 0,                !Constant to set a flag
;         1638      enaBE = 1,                  !Enable mlmr register function
;         1639      DISABE = 0;                 !Disable mlmr register function
;         1640
```

```
;          1641  %sbttl 'FIELD DECLARATIONS'
;          1642  !
;          1643  ! FIELD DECLARATIONS
;          1644  !
;          1645
;          1646  field
;          1647      !
;          1648      ! REMAINING ERROR TABLE STRUCTURE MAP
;          1649      !
;          1650      MAP_REM_TBL =
;          1651          set
;          1652          ROW = [8, 8, 0],                    !Select failing row number position
;          1653          COL = [0, 8, 0],                    !Select failing column position
;          1654          RO_CL = [0, 16, 0]                  !Select both row and column position
;          1655          tes,
;          1656      !
;          1657      ! FAILING CHIP AND PATTERN STRUCTURE MAP
;          1658      !
;          1659      MAP_CHIP_TBL =
;          1660          set
;          1661          FAULT = [15, 1, 0],                 !Set when this chip has additional failing row/col
;          1662          PAT_0 = [14, 1, 0],                 !This chip fails zeroes pattern
;          1663          PAT_1 = [13, 1, 0],                 !This chip fails ones pattern
;          1664          RAN_1 = [12, 1, 0],                 !This chip fails random pat 1
;          1665          RAN_2 = [11, 1, 0],                 !This chip fails random pat 2
;          1666          RAN_3 = [10, 1, 0],                 !This chip fails random pat 3
;          1667          RAN_4 = [9, 1, 0],                  !This chip fails random pat 4
;          1668          RAN_5 = [8, 1, 0],                  !This chip fails random pat 5
;          1669          RAN_6 = [7, 1, 0],                  !This chip fails random data 6
;          1670          RAN_7 = [6, 1, 0],                  !This chip fails random data 7
;          1671          RAN_8 = [5, 1, 0],                  !This chip fails random data 8
;          1672          RAN_9 = [4, 1, 0],                  !This chip fails random data 9
;          1673          RAN_10 = [3, 1, 0],                 !This chip fails random data 10
;          1674          RAN_11 = [2, 1, 0],                 !This chip fails random data 11
;          1675          RAN_12 = [1, 1, 0],                 !This chip fails random data 12
;          1676          RAN_13 = [0, 1, 0],                 !This chip fails random data 13
;          1677          PATS = [0, 15, 0],                  !Select all failing patterns
;          1678          ALL = [0, 16, 0]                    !Select all bits in byte
;          1679          tes,
;          1680      !
;          1681      ! WRITE BUFFER STRUCTURE MAP
;          1682      !
;          1683      MAP_WRT_BUF =
;          1684          set
;          1685          BIT_0 = [0, 1, 0],                  !Defines bit 0 of write buffer
;          1686          BIT_1 = [1, 1, 0],                  !Defines bit 1 "    "     "
;          1687          BIT_2 = [2, 1, 0],                  !Defines bit 2 "    "     "
;          1688          BIT_3 = [3, 1, 0],                  !Defines bit 3 "    "     "
;          1689          BIT_4 = [4, 1, 0],                  !Defines bit 4 "    "     "
;          1690          BIT_5 = [5, 1, 0],                  !Defines bit 5 "    "     "
;          1691          BIT_6 = [6, 1, 0],                  !Defines bit 6 "    "     "
;          1692          BIT_7 = [7, 1, 0],                  !Defines bit 7 "    "     "
```

```
1693          BIT_8 = [8, 1, 0],                    !Defines bit 8 "   "   "
1694          BIT_9 = [9, 1, 0],                    !Defines bit 9 "   "   "
1695          BIT_10 = [10, 1, 0],                  !Defines bit 10 "   "   "
1696          BIT_11 = [11, 1, 0],                  !Defines bit 11 "   "   "
1697          BIT_12 = [12, 1, 0],                  !Defines bit 12 "   "   "
1698          BIT_13 = [13, 1, 0],                  !Defines bit 13 "   "   "
1699          BIT_14 = [14, 1, 0],                  !Defines bit 14 "   "   "
1700          BIT_15 = [15, 1, 0],                  !Defines bit 15 "   "   "
1701          WRD = [0, 16, 0]                      !Defines word of the write buffer
1702          tes,
1703      !
1704      ! TEMPORARY BLAST TABLE STRUCTURE MAP
1705      !
1706      MAP_TMP_BLST_TBL =
1707          set
1708          NIB_NO = [0, 4, 0],                   !Defines failing chips nibble position
1709          R_C_NO = [4, 8, 0],                   !Defines chips failing row or column
1710          R_C = [15, 1, 0]                      !Set = row address, clr = column address
1711          tes,
1712      !
1713      ! RH / ML-11 REGISTER ACCESSING STRUCTRUE MAP
1714      !
1715      MAP_ML11_REG =
1716          set
1717          !
1718          ! MLCS1 CONTROL STAUTS REGISTER 1
1719          !
1720          sc = [15, 1, 0],                      !Special condition
1721          TRE = [14, 1, 0],                     !Transfer error
1722          MCPE = [13, 1, 0],                    !Bus parity error
1723          DVA = [11, 1, 0],                     !Drive available
1724          RDY = [7, 1, 0],                      !Drive ready
1725          IE = [6, 1, 0],                       !Interrupt enable
1726          GO = [0, 1, 0],                       !Funtion go bit
1727          FUNC = [0, 6, 0],                     !Funtion code
1728          !
1729          ! MLWC WORD COUNT REGISTER
1730          !
1731          wc_reg = [0, 16, 0],                  !Word count register
1732          !
1733          ! MLBA UNIBUS ADDRESS REGISTER
1734          !
1735          BA_REG = [0, 16, 0],                  !Unibus address register
1736          !
1737          ! MLDA DESIRED ADDRESS REGISTER
1738          !
1739          da_reg = [0, 16, 0],                  !desired sector register
1740          !
1741          ! MLCS2 CONTROL AND STATUS REGISTER 2
1742          !
1743          DLT = [15, 1, 0],                     !Data late
1744          WCE = [14, 1, 0],                     !Write check error
```

```
1745            PE = [13, 1, 0],                    !Unibus parity error
1746            NED = [12, 1, 0],                   !Non-existent drive
1747            NEM = [11, 1, 0],                   !Non-existent memory
1748            PGE = [10, 1, 0],                   !Program error
1749            MXF = [9, 1, 0],                    !Missed tranfer
1750            MDPE = [8, 1, 0],                   !Massbus data bus parity error
1751            ORDY = [7, 1, 0],                   !Output ready
1752            IRDY = [6, 1, 0],                   !Input ready
1753            CLR = [5, 1, 0],                    !Controller clear
1754            PAT = [4, 0],                       !Parity test
1755            BAI = [3, 1, 0],                    !Unibus address increment inhibit
1756            DRV_SEL = [0, 3, 0],                !Unit select
1757            !
1758            ! MLDS DRIVE STATUS REGISTER
1759            !
1760            ATTN = [15, 1, 0],                  !Attention active
1761            COMP_ERR = [14, 1, 0],              !Error summary
1762            MOL = [12, 1, 0],                   !Medium on line
1763            LBT = [10, 1, 0],                   !Last block
1764            DPR = [8, 1, 0],                    !Drive present
1765            DRY = [7, 1, 0],                    !Drive ready
1766            VV = [6, 1, 0],                     !Volume valid
1767            !
1768            ! MLER ERROR REGISTER
1769            !
1770            DCK = [15, 1, 0],                   !Data check
1771            UNS = [14, 1, 0],                   !Drive unsafe
1772            OPI = [13, 1, 0],                   !Operation incomplete
1773            IAE = [10, 1, 0],                   !Invalid address error
1774            AOE = [9, 1, 0],                    !Address overflow error
1775            ECH_ERR = [6, 1, 0],                !Ecc hard error
1776            DPAR = [5, 1, 0],                   !Data parity error
1777            CPAR = [3, 1, 0],                   !Control parity error
1778            RMR = [2, 1, 0],                    !Register mod erfused error
1779            ILR = [1, 1, 0],                    !Illegal register error
1780            ILF = [0, 1, 0],                    !Illegal function
1781            !
1782            !  MLAS ATTENTION SUMMARY REGISTER
1783            !
1784            attn_reg = [0, 16, 0],              !attention summary register
1785            !
1786            ! MLPA PROM ADDRESS REGISTER
1787            !
1788            PA_REG = [0, 16, 0],                !Prom address register
1789            !
1790            ! MLMR MAINTENANCE REGISTER
1791            !
1792            SIZING = [11, 5, 0],                !System sizing
1793            TRT = [8, 2, 0],                    !Transfer rate
1794            ARR_TYP = [10, 1, 0],               !Array type
1795            REF_MAR = [7, 1, 0],                !Refresh margin
1796            P_RW = [6, 1, 0],                   !Prom read/write
```

```
1797            P_DIS = [5, 1, 0],                  !Prom disable
1798            D_CLK = [4, 1, 0],                  !Data clock
1799            D_DM = [3, 1, 0],                   !Data diag mode
1800            D_EN = [2, 1, 0],                   !Data check enable
1801            E_DIS = [1, 1, 0],                  !Ecc disable mode
1802            E_DM = [0, 1, 0],                   !Ecc diag mode
1803            MR_REG = [0, 16, 0],                !Maintenance register
1804            !
1805            ! MLDT DRIVE TYPE REGISTER
1806            !
1807            DRVTYP = [1, 8, 0],                 !Drive type
1808            DT_REG = [0, 1, 0],                 !Drive type register
1809            !
1810            ! MLSN SERIAL NUMBER REGISTER
1811            !
1812            SERIAL_REG = [0, 16, 0],            !Serial number register
1813            !
1814            ! MLE1 ECC CRC WORD REGISTER 1
1815            !
1816            PAR_CRC_WRD = [8, 6, 0],            !Parity crc word
1817            CRC$A = [0, 6, 0],                  !CRC word A-A
1818            E1_REG = [0, 16, 0],                !Ecc crc register 1
1819            !
1820            ! MLE2 ECC CRC WORD REGISTER 2
1821            !
1822            B_32 = [8, 1, 0],                   !Data bit 32
1823            B_33 = [9, 1, 0],                   !Data bit 33
1824            B_34 = [10, 1, 0],                  !Data bit 34
1825            B_35 = [11, 1, 0],                  !Data bit 35
1826            B_36 = [12, 1, 0],                  !Data bit 36
1827            B_37 = [13, 1, 0],                  !Data bit 37
1828            B_38 = [14, 1, 0],                  !Data bit 38
1829            CRC$B = [0, 6, 0],                  !Crc word b-b
1830            E2_REG = [0, 16, 0],                !Ecc crc register 2
1831            !
1832            ! MLD1 DATA DIAGNOSTIC REGISTER 1
1833            !
1834            b_0 = [0, 1, 0],                    !data bit 0
1835            B_1 = [1, 1, 0],                    !Data bit 1
1836            B_2 = [2, 1, 0],                    !Data bit 2
1837            B_3 = [3, 1, 0],                    !Data bit 3
1838            B_4 = [4, 1, 0],                    !Data bit 4
1839            B_5 = [5, 1, 0],                    !Data bit 5
1840            B_6 = [6, 1, 0],                    !Data bit 6
1841            B_7 = [7, 1, 0],                    !Data bit 7
1842            B_8 = [8, 1, 0],                    !Data bit 8
1843            B_9 = [9, 1, 0],                    !Data bit 9
1844            B_10 = [10, 1, 0],                  !Data bit 10
1845            B_11 = [11, 1, 0],                  !Data bit 11
1846            B_12 = [12, 1, 0],                  !Data bit 12
1847            B_13 = [13, 1, 0],                  !Data bit 13
1848            B_14 = [14, 1, 0],                  !Data bit 14
```

```
1849          B_15 = [15, 1, 0],                    !Data bit 15
1850          DB1_REG = [0, 16, 0],                 !Data diag register 1
1851          !
1852          ! MLD2 DATA DIAGNOSTIC REGISTER 2
1853          !
1854          B_16 = [0, 15, 0],                    !Data bit 16
1855          B_17 = [1, 1, 0],                     !Data bit 17
1856          B_18 = [2, 1, 0],                     !Data bit 18
1857          B_19 = [3, 1, 0],                     !Data bit 19
1858          B_20 = [4, 1, 0],                     !Data bit 20
1859          B_21 = [5, 1, 0],                     !Data bit 21
1860          B_22 = [6, 1, 0],                     !Data bit 22
1861          B_23 = [7, 1, 0],                     !Data bit 23
1862          B_24 = [8, 1, 0],                     !Data bit 24
1863          B_25 = [9, 1, 0],                     !Data bit 25
1864          B_26 = [10, 1, 0],                    !Data bit 26
1865          B_27 = [11, 1, 0],                    !Data bit 27
1866          B_28 = [12, 1, 0],                    !Data bit 28
1867          B_29 = [13, 1, 0],                    !Data bit 29
1868          B_30 = [14, 1, 0],                    !Data bit 30
1869          B_31 = [15, 1, 0],                    !Data bit 31
1870          DB2_REG = [0, 16, 0],                 !Data diag register 2
1871          !
1872          ! MLEE ECC ERROR REGISTER
1873          !
1874          UNC = [15, 1, 0],                     !Uncorrectable error
1875          SGL = [14, 1, 0],                     !Single error
1876          crc = [13, 1, 0],                     !Crc error
1877          CHAN = [6, 6, 0],                     !Channel in error
1878          E_Bit = [0, 6, 0],                    !Error function
1879          !
1880          ! MLEL ECC ERROR LOCATION REGISTER
1881          !
1882          EL_REG = [0, 16, 0],                  !Error location register
1883          !
1884          ! MLPD PROM DATA REGISTER
1885          !
1886          PD_REG = [0, 16, 0],                  !Prom data register
1887          !
1888          ! MLBAE BUS ADDRESS EXTENTION REGISTER
1889          !
1890          BAE_REG = [0, 16, 0],                 !Bus address extention register
1891          !
1892          ! MLCS3 CONTROL AND STATUS REGISTER 3
1893          !
1894          CS3_REG = [0, 16, 0],                 !Control and status register 3
1895          !
1896          ! ML_ALL ACCESS ALL BITS IN SELECTED REGISTER
1897          !
1898          ML_ALL = [0, 16, 0]                   !Access all register bits
1899          tes,
1900      !
```

```
;          1901       ! Ver czmlcb prom maintenance error
;          1902       ! log table error log table map
;          1903       !
;          1904       PM_MAP =
;          1905           set
;          1906           BNK$_PM = [0, 0, 2, 0],
;          1907           BRD$_PM = [0, 2, 4, 0],
;          1908           BIT$_PM = [0, 6, 7, 0],
;          1909           UNIT$_PM = [0, 13, 3, 0],
;          1910           SUM$_PM = [1, 0, 16, 0],
;          1911           WRD$_0 = [0, 0, 16, 0],
;          1912           WRD$_1 = [1, 0, 16, 0]
;          1913           tes;
;          1914
```

```
;          1915  %sbttl 'STRUCTURE DECLARATIONS'
;          1916  !
;          1917  ! STRUCTURE DECLARATIONS
;          1918  !
;          1919
;          1920  structure
;          1921      !
;          1922      ! ML-11 register accessing structure
;          1923      !
;          1924      ! this structure allows ML-11 register
;          1925      ! accessing to be transportable between
;          1926      ! the PDP-11 and VAX diagnostic supervisors
;          1927      !
;          1928  !<BLF/NOFORMAT>
;          1929
;          1930
;          1931      RH [O, P, S, E] =
;          1932          begin
;          1933
;          1934          local
;          1935              ML_REG;
;          1936
;          1937          ML_REG = .(RH + %upval*0)<0, %bpval, 0>;
;          1938          ML_REG
;          1939          end <P, S, E>;
;          1940
;          1941  !<BLF/FORMAT>
```

```
;          1942  %sbttl 'EXTERNAL DECLARATIONS'
;          1943  !
;          1944  ! EXTERNAL DECLARATIONS
;          1945  !
;          1946
;          1947  external routine
;          1948      !
;          1949      ! Random number generator routine written in macro source code
;          1950      !
;          1951      RANGEN : novalue;
;          1952
;          1953  external
;          1954      !
;          1955      ! Random number generator routine seed variables
;          1956      !
;          1957      SEED1,                                  !First word of random number buffer
;          1958      SEED2,                                  !Second word of random number buffer
;          1959      SEED3,                                  !Third word of random number buffer
;          1960      RANDAT,                                 !Linkage which random numbers are returned
;          1961      !
;          1962      ! Diagnostic supervisor global variables
;          1963      !
;          1964      L$UNIT;                                 !Supervisor storage of units selected
;          1965
```

```
;          1966  %sbttl 'OWN STORAGE DECLARATIONS'
;          1967  !
;          1968  ! Own storage declarations
;          1969  !
;          1970
;          1971  own
;          1972      !
;          1973      ! VER CZMLCB added these storage structures to this version
;          1974      !
;          1975      ! Prom Maintenance error summary report structures
;          1976      !
;          1977      PM_COUNT,                                   !Counts number of bad chips found
;          1978      PM_LOG : blockvector [128, 2, word] field (PM_MAP),
;          1979      !
;          1980      ! Error map interigation structures
;          1981      !
;          1982      COL_CNT_TBL : vector [256, byte] volatile,  !Column count table
;          1983      REM_TBL : block [100] field (MAP_REM_TBL) volatile, !Remainder table
;          1984      COPIED_REM_TBL : block [100] field (MAP_REM_TBL) volatile,  !Copied remainder table
;          1985      COL_PTR : vector [10, byte] volatile,       !Column count table pointer
;          1986      !
;          1987      ! I/O buffers
;          1988      !
;          1989      ! VER CZMLCB changed 128 to 256
;          1990      !
;          1991      WRT_BUF : block [(256 + 511*2)] field (MAP_WRT_BUF) volatile,       !Write buffer
;          1992      RD_BUF : vector [256] volatile,             !Read buffer
;          1993      !
;          1994      ! Failing row and sector storage structure
;          1995      !
;          1996      ERROR_MAP : blockvector [512, 8] volatile,  !Error map of failing rows and sectors
;          1997      !
;          1998      ! Failing chips and pattern storage structure
;          1999      !
;          2000      CHIP_TBL : block [39] field (MAP_CHIP_TBL) volatile,        !Failing chip table
;          2001      !
;          2002      ! Row & col blast information storage structure
;          2003      !
;          2004      TMP_BLST_TBL : block [10] field (MAP_TMP_BLST_TBL) volatile,        !Temporary blast table
;          2005      BLAST_TBL : blockvector [4, 512] volatile,  !Actual blast table
;          2006      COL_BASE,                                   !Column adrs base index into blast table
;          2007      !
;          2008      ! ML-11 accessing structure referance
;          2009      !
;          2010      ML_ADDR : ref RH field (MAP_ML11_REG) volatile,     !Rh base register address
;          2011      !
;          2012      ! Init code global variables
;          2013      !
;          2014      MAX_CHIP_COL,                               !Number of columns per mos ram
;          2015      ML_DUT : volatile,                          !ML-11 divice under test
;          2016      ARRSBNK_SEL : volatile,             !Fabricated desired sector adrs where array maint is performed
;          2017      TSTED_BNK : volatile,                       !Count of how many banks to array maint
```

```
;          2018      INC_BNK : volatile,                     !Bank adrs increment value
;          2019      INC_ARR : volatile,                     !Array adrs increment value
;          2020      ARR_SEL_POS : volatile,                 !Field select  position expression
;          2021      BNK_SEL_POS : volatile,                 !Field select position expression
;          2022      BNK_NUM_SEC : volatile,                 !Indicates number of sectors per bank. 16k = 128, 64k = 256
;          2023      !
;          2024      ! Program flag registers
;          2025      !
;          2026      BAD_BNK_REG : bitvector [8] volatile,   !Indicates if bank has new errors
;          2027      FLG_REG : bitvector [16] volatile,      !General flag register
;          2028      DEGRADE_MOD_REG : bitvector [8] volatile,   !Degrade array mode flag register
;          2029      !
;          2030      ! Miscellaneous variables
;          2031      !
;          2032      SIZE : volatile,                        !Transfers word count size
;          2033      DST : volatile,                         !Transfers destination address
;          2034      SRC : volatile,                         !Transfers source address
;          2035      LST_TSTED_ARR : volatile,               !Last array module number in ml-11 system
;          2036      RAND_PASS : volatile;                   !Random data pass variable
;          2037
;          2038  !
;          2039  ! Supervisor global equates
;          2040  !
;          2041  EQUALS;
```

```
;          2042  %sbttl 'BIND DECLARATIONS'
;          2043  !
;          2044  ! Bind declarations
;          2045  !
;          2046
;          2047  bind
;          2048      !
;          2049      ! VER CZMLBB newly defined messages to print out statisical
;          2050      ! prom maintenance messages.
;          2051      !
;          2052      FMT16 = uplit (%asciz'%N%D1%A(D)      %D2%A(D)      %D1%A(D)      %D2%A(D)      %D6%A(D)'),
;          2053      PM$HEADER = uplit (%asciz'UNIT #:   ARRAY #:   BANK #:   BIT #:      COUNT #:'),
;          2054      PM_HEADER = uplit (%asciz'PROM MAINTENANCE PERFORMANCE SUMMARY REPORT'),
;          2055      PM_14_MSG = uplit (%asciz'NO ADDITIONAL ERRORS DETECTED WITHIN THIS UNIT'),
;          2056      PM_10_MSG = uplit (%asciz'%N%A                THE FOLLOWING LISTS AN ACCUMLATIVE COUNT OF PREVIOUSLY AND NEWLY'),
;          2057      PM_11_MSG = uplit (%asciz'%N%A                FAILING  ROWS AND COLUMNS DETECTED  WITHIN EACH  ERRORING  CHIP.'),
;          2058      PM_12_MSG = uplit (%asciz'%N%A                HOWEVER THIS LIST DOES NOT NECESSARILY INDICATE THE NUMBER OF NEWLY'
;          2059      PM_13_MSG = uplit (%asciz'%N%A                FAILING ROWS OR COLUMNS BLASTED%N'),
;          2060      !
;          2061      ! Error and run time messages
;          2062      !
;          2063      ILL_CMD_MSG = uplit (%asciz'ILLEGAL PROM MAINTENANCE PROGRAM COMMAND '),
;          2064      BGN_MSG = uplit (%asciz'STARTING PROM MAINTENANCE'),
;          2065      START_MSG = uplit (%asciz'TYPE START TO EXECUTE PROGRAM'),
;          2066      HQ_ERR_MSG = uplit (%asciz'HARDWARE QUESTIONS NOT ANSWERED CORRECTLY '),
;          2067      HQ_MSG = uplit (%asciz'BLASTING SYSTEM, ARRAY OR BANK NOT SELECTED '),
;          2068      END_MSG = uplit (%asciz'PROM MAINTANENCE COMPLETED '),
;          2069      SUPRES_MSG = uplit (%asciz'SUPPRESSING PROGRAM EXECUTION '),
;          2070      RET_DRS_MSG = uplit (%asciz'RETURNING TO DRS>'),
;          2071      LUN_MISS_MSG = uplit (%asciz'LUN 0 P-TABLE NOT PRESENT'),
;          2072      CON_A_MSG = uplit (%asciz'CONDITION A'),
;          2073      CON_B_MSG = uplit (%asciz'CONDITION B'),
;          2074      CON_C_MSG = uplit (%asciz'CONDITION C'),
;          2075      CON_D_MSG = uplit (%asciz'CONDITION D '),
;          2076      UNC_CHIP_MSG = uplit (%asciz'UNCONFIRMED FAILING CHIP'),
;          2077      GTR_MSG = uplit (%asciz'MORE THAN ONE UNIT SELECTED FOR PROM MAINT'),
;          2078      UNIT_SEL_MSG = uplit (%asciz'ALL UNITS EXCEPT UNIT 0 WILL BE IGNORED'),
;          2079      SW_BUG_MSG = uplit (%asciz'UNEXPECTED SOFTWARE BUG DETECTED NOTIFY DIAG ENG'),
;          2080      ABORT_MSG = uplit (%asciz'PROM MAINTANENCE ABORTED FOR THIS ARRAY'),
;          2081      NO_AD_MSG = uplit (%asciz'NO ADDITIONAL ERRORS BLASTED FOR THIS ARRAY'),
;          2082      UNC_ERR_MSG = uplit (%asciz'UNCORRECTABLE ERRORS DETECTED AFTER BLASTING'),
;          2083      TIME_OUT_MSG = uplit (%asciz'DATA CLOCK BIT FAILED TO RESET AFTER PROM WRITE'),
;          2084      REP_M7363_MSG = uplit (%asciz'REPLACE ARRAY DATA MODULE M7363'),
;          2085      MEM_ERR_MSG = uplit (%asciz'MEMORY ERRORS STILL EXIST AFTER BLASTING'),
;          2086      UNX_DRV_ERR_MSG = uplit (%asciz'UNEXPECTED DRIVE ERRORS DETECTED'),
;          2087      RD_XFER_MSG = uplit (%asciz'OCCURED DURING A MASS BUS READ TRANSFER'),
;          2088      WRT_XFER_MSG = uplit (%asciz'OCCURED DURING A MASS BUS WRITE TRANSFER'),
;          2089      WT_CHK_XFER_MSG = uplit (%asciz'OCCURED DURING A MASS BUS WRITE CHECK TRANSFER'),
;          2090      X_MSG = uplit (%asciz''),
;          2091      !
;          2092      ! Printing formats
;          2093      !
```

```
;          2094      ONE_MSG = uplit (%asciz'%T%N'),                   ! PRINT ONE MESSAGE
;          2095      TWO_MSG = uplit (%asciz'%T%T%N'),                 !PRINT TWO MESSAGES
;          2096      THREE_MSG = uplit (%asciz'%T%T%T%N'),             ! PRINT THREE MESAGES
;          2097      FOUR_MSG = uplit (%asciz'%T%T%T%T%N'),            ! PRINT FOUR MESSAGES
;          2098      CRLF = uplit (%asciz'%N'),
;          2099      LF = uplit (%asciz'%N'),                          !PRINT A LINE FEED <CR>
;          2100      LFS = uplit (%asciz'%N%N'),                       ! PRINT TWO <CR>
;          2101      O_1_PRINT = uplit (%asciz'%T%O1%N'),              ! PRINT OCTAL DIGIT AND A MESSAGE
;          2102      O_6_PRINT = uplit (%asciz'%T%O6%N'),              ! PRINT SIX OCTAL DIGITS AND A MESSAGE
;          2103      D_2_PRINT = uplit (%asciz'%T%D2%N'),              ! PRINT A TWO DIGIT NUMBER AND A MESSAGE
;          2104      DRV_SEL_PRINT = uplit (%asciz'%ADRIVE %O1%A SELECTED%N'),
;          2105                                                        ! PRINT THE DRIVE SELECTED NUMBER TO THE OPERATOR
;          2106      ARR_SEL_PRINT = uplit (%asciz'%S%S%S%S%S%S%S%S%S%ATESTING ARRAY %D2%N'),
;          2107                                                        ! PRINT THE ARRAY SELECTED TO THE OPERATOR
;          2108      BNK_SEL_PRINT = uplit (%asciz'%S%S%S%S%S%S%S%S%S%S%S%S%S%S%S%S%S%S%ABANK%S%S%D1%N'),
;          2109                                                        ! PRINT THE BANK PRESENTLY UNDER TEST TO THE OPERATOR
;          2110      A_B_C_PRINT = uplit (%asciz'%AARRAY%S%D2%S%ABANK%S%O1%S%ABIT #%S%D2%N'),
;          2111      A_B_N_PRINT = uplit (%asciz'%AARRAY%S%D2%S%ABANK%S%O1%S%ANIBBLE%S%D2%N'),
;          2112      REP_PRINT = uplit (%asciz'%AREPLACE ARRAY MODULE%S%D2%S%S%AOR RERUN PROGRAM%N'),
;          2113      A_B_PRINT = uplit (%asciz'%AARRAY%S%D2%S%ABANK%S%O1%N'),
;          2114      CS1_PRINT = uplit (%asciz'%S%S%S%S%AMLCS1%S%S%S%S%S%S%S%S%S%S%S%S%S%S%S%S%S%O6%N'),
;          2115      WC_PRINT = uplit (%asciz'%S%S%S%S%AMLWC%S%S%S%S%S%S%S%S%S%S%S%S%S%S%S%S%S%O6%N'),
;          2116      BA_PRINT = uplit (%asciz'%S%S%S%S%AMLBA%S%S%S%S%S%S%S%S%S%S%S%S%S%S%S%S%S%O6%N'),
;          2117      DA_PRINT = uplit (%asciz'%S%S%S%S%AMLDA%S%S%S%S%S%S%S%S%S%S%S%S%S%S%S%S%S%O6%N'),
;          2118      CS2_PRINT = uplit (%asciz'%S%S%S%S%AMLCS2%S%S%S%S%S%S%S%S%S%S%S%S%S%S%S%S%S%O6%N'),
;          2119      DS_PRINT = uplit (%asciz'%S%S%S%S%AMLDS%S%S%S%S%S%S%S%S%S%S%S%S%S%S%S%S%S%O6%N'),
;          2120      ER_PRINT = uplit (%asciz'%S%S%S%S%AMLER%S%S%S%S%S%S%S%S%S%S%S%S%S%S%S%S%S%O6%N'),
;          2121      AS_PRINT = uplit (%asciz'%S%S%S%S%AMLAS%S%S%S%S%S%S%S%S%S%S%S%S%S%S%S%S%S%O6%N'),
;          2122      MR_PRINT = uplit (%asciz'%S%S%S%S%AMLMR%S%S%S%S%S%S%S%S%S%S%S%S%S%S%S%S%S%O6%N'),
;          2123      DT_PRINT = uplit (%asciz'%S%S%S%S%AMLDT%S%S%S%S%S%S%S%S%S%S%S%S%S%S%S%S%S%O6%N'),
;          2124      SN_PRINT = uplit (%asciz'%S%S%S%S%AMLSN%S%S%S%S%S%S%S%S%S%S%S%S%S%S%S%S%S%O6%N'),
;          2125      EE_PRINT = uplit (%asciz'%S%S%S%S%AMLEE%S%S%S%S%S%S%S%S%S%S%S%S%S%S%S%S%S%O6%N'),
;          2126      EL_PRINT = uplit (%asciz'%S%S%S%S%AMLEL%S%S%S%S%S%S%S%S%S%S%S%S%S%S%S%S%S%O6%N'),
;          2127      HEADER_PRINT = uplit (%asciz'%S%S%S%S%S%S%S%S%S%S%AREGISTER  DUMP%N%N'),
;          2128      COL_DESC_PRINT = uplit (%asciz'%S%S%AREGISTER NAME%S%S%S%S%S%S%ACONTENTS%N'),
;          2129      X_PRINT = uplit (%asciz'');
;          2130
```

```
                2131  %sbttl 'MACRO DECLARATIONS'
                2132  !
                2133  ! Macro declarations
                2134  !
                2135
                2136  macro
                2137      !
                2138      ! ML-LL register accessing macro
                2139      !
                2140      ! this macro allows ml-11 register accessing to be
                2141      ! transportable between the PDP-11 and VAX diagnostic
                2142      ! supervisors.
                2143      !
            M   2144      WRT_RH (O, FIELDNAM, IMAGE) =
            M   2145              begin
            M   2146
            M   2147              local
            M   2148                  MLREG;
            M   2149
            M   2150              MLREG = .ML_ADDR [O, ML_ALL];
            M   2151              MLREG<%fieldexpand (FIELDNAM)> = IMAGE;
            M   2152              (.ML_ADDR + %upval*O) = .MLREG;
                2153              end;%,
                2154      !
                2155      ! Clear rh and mass bus devices
                2156      !
            M   2157      CLR_MBUS =
            M   2158  WRT_RH (MLCS2, CLR,  ONE);!Clear the mass bus
                2159  WRT_RH (MLCS2, DRV_SEL,  .ML_DUT);%,          !Restore the drive select number
                2160  !
                2161      ! Clear mass bus devices
                2162      !
            M   2163      CLR_DRIVE =
                2164  WRT_RH (MLCS1, FUNC,  FUNC_2);%,              !Clear the drive
                2165  !
                2166      ! Maintanence register diag mode setting macros
                2167      !
            M   2168      RD_PROM_MODE =
                2169  WRT_RH(MLMR,MR_REG,PM_RD_MODE);%,             !Enable prom reads
            M   2170      WRT_PROM_MODE =
                2171  WRT_RH(MLMR,MR_REG,PM_WRT_MODE);%,            !Enable prom writes
            M   2172      DAT_DM (X) =
                2173  WRT_RH (MLMR,D_DM,  X);%,                     !Enable/disable data diag mode
            M   2174      ECC_DIS (X) =
                2175  WRT_RH (MLMR,E_DIS,  X);%,                    !Enable/disable ecc disable mode
            M   2176      PROM_RW (X) =
                2177  WRT_RH (MLMR,P_RW,  X);%,                     !Enable prom read write
            M   2178      DAT_CLK =
                2179  WRT_RH (MLMR,D_CLK,  ONE);%,                  !Data clock
            M   2180      PROM_DIS (X) =
                2181  WRT_RH (MLMR,P_DIS,  X);%,                    !Enable prom disable
            M   2182      DCK_EN (X) =
```

```
;          2183  WRT_RH (MLMR,D_EN,  X);%,                !Enable data check enable
;        M 2184      ECC_DM (X) =
;          2185  WRT_RH (MLMR,E_DM,  X);%,                !Enable ecc disable
;          2186  !
;          2187      ! Miscellanious  macros definitions
;          2188      !
;        M 2189      PM_RD_MODE =
;          2190  %o'000040'%,                             !Enable prom read mode
;        M 2191      PM_WRT_MODE =
;          2192  %o'000T40'%,                             !Enable prom write mode
;        M 2193      FULL_WRD =
;          2194  0, 16, 0%;                               !Selects full word of data element
;          2195
```

```
;          2196  %sbttl 'EXTENDED MESSAGE PRINTING SECTION'
;          2197
;          2198  !++
;          2199  !  Functional description: register dump
;          2200  !                           This bgnmsg will get called when the
;          2201  !                           drive detects errors during a mass
;          2202  !                           bus transfer.  this bgnmsg will print
;          2203  !                           out to the terminal the error registers.
;          2204  !--
;          2205
;          2206  BGNMSG (DUMPER);                                  !Print the dump message

                                          .TITLE  BSKEL4
                                          .IDENT  /REV B /

000000                                    .PSECT  $PLIT$,  D
000000     045     116     045    P.AAA:  .ASCII  /%N%/
000003     104     061     045            .ASCII  /D1%/
000006     101     050     104            .ASCII  /A(D/
000011     051     040     040            .ASCII  /)   /
000014     040     040     040            .ASCII  /    /
000017     040     045     104            .ASCII  / %D/
000022     062     045     101            .ASCII  /2%A/
000025     050     104     051            .ASCII  /(D)/
000030     040     040     040            .ASCII  /    /
000033     040     040     040            .ASCII  /    /
000036     045     104     061            .ASCII  /%D1/
000041     045     101     050            .ASCII  /%A(/
000044     104     051     040            .ASCII  /D) /
000047     040     040     040            .ASCII  /    /
000052     040     040     045            .ASCII  /   %/
000055     104     062     045            .ASCII  /D2%/
000060     101     050     104            .ASCII  /A(D/
000063     051     040     040            .ASCII  /)   /
000066     040     040     040            .ASCII  /    /
000071     040     040     045            .ASCII  /   %/
000074     104     066     045            .ASCII  /D6%/
000077     101     050     104            .ASCII  /A(D/
000102     051     000                    .ASCII  /)/<00>
000104     125     116     111    P.AAB:  .ASCII  /UNI/
000107     124     040     043            .ASCII  /T #/
000112     072     040     040            .ASCII  /:   /
000115     040     101     122            .ASCII  / AR/
000120     122     101     131            .ASCII  /RAY/
000123     040     043     072            .ASCII  / #:/
000126     040     040     040            .ASCII  /    /
000131     102     101     116            .ASCII  /BAN/
000134     113     040     043            .ASCII  /K #/
000137     072     040     040            .ASCII  /:   /
000142     040     102     111            .ASCII  / BI/
```

```
000145  124  040  043                .ASCII  /T #/
000150  072  040  040                .ASCII  /:  /
000153  040  040  040                .ASCII  /   /
000156  040  103  117                .ASCII  / CO/
000161  125  116  124                .ASCII  /UNT/
000164  040  043  072                .ASCII  / #:/
000167  000                         .ASCII  <00>
000170  120  122  117      P.AAC:    .ASCII  /PRO/
000173  115  040  115                .ASCII  /M M/
000176  101  111  116                .ASCII  /AIN/
000201  124  105  116                .ASCII  /TEN/
000204  101  116  103                .ASCII  /ANC/
000207  105  040  120                .ASCII  /E P/
000212  105  122  106                .ASCII  /ERF/
000215  117  122  115                .ASCII  /ORM/
000220  101  116  103                .ASCII  /ANC/
000223  105  040  123                .ASCII  /E S/
000226  125  115  115                .ASCII  /UMM/
000231  101  122  131                .ASCII  /ARY/
000234  040  122  105                .ASCII  / RE/
000237  120  117  122                .ASCII  /POR/
000242  124  000                    .ASCII  /T/<00>
000244  116  117  040      P.AAD:    .ASCII  /NO /
000247  101  104  104                .ASCII  /ADD/
000252  111  124  111                .ASCII  /ITI/
000255  117  116  101                .ASCII  /ONA/
000260  114  040  105                .ASCII  /L E/
000263  122  122  117                .ASCII  /RRO/
000266  122  123  040                .ASCII  /RS /
000271  104  105  124                .ASCII  /DET/
000274  105  103  124                .ASCII  /ECT/
000277  105  104  040                .ASCII  /ED /
000302  127  111  124                .ASCII  /WIT/
000305  110  111  116                .ASCII  /HIN/
000310  040  124  110                .ASCII  / TH/
000313  111  123  040                .ASCII  /IS /
000316  125  116  111                .ASCII  /UNI/
000321  124  000  000                .ASCII  /T/<00><00>
000324  045  116  045      P.AAE:    .ASCII  /%N%/
000327  101  011  011                .ASCII  /A/<11><11>
000332  124  110  105                .ASCII  /THE/
000335  040  106  117                .ASCII  / FO/
000340  114  114  117                .ASCII  /LLO/
000343  127  111  116                .ASCII  /WIN/
000346  107  040  114                .ASCII  /G L/
000351  111  123  124                .ASCII  /IST/
000354  123  040  101                .ASCII  /S A/
000357  116  040  101                .ASCII  /N A/
000362  103  103  125                .ASCII  /CCU/
000365  115  114  101                .ASCII  /MLA/
000370  124  111  126                .ASCII  /TIV/
000373  105  040  103                .ASCII  /E C/
```

```
000376  117  125  116                .ASCII  /OUN/
000401  124  040  117                .ASCII  /T O/
000404  106  040  120                .ASCII  /F P/
000407  122  105  126                .ASCII  /REV/
000412  111  117  125                .ASCII  /IOU/
000415  123  114  131                .ASCII  /SLY/
000420  040  101  116                .ASCII  / AN/
000423  104  040  116                .ASCII  /D N/
000426  105  127  114                .ASCII  /EWL/
000431  131  000  000                .ASCII  /Y/<00><00>
000434  045  116  045          P.AAF:  .ASCII  /%N%/
000437  101  011  011                .ASCII  /A/<11><11>
000442  106  101  111                .ASCII  /FAI/
000445  114  111  116                .ASCII  /LIN/
000450  107  040  040                .ASCII  /G  /
000453  122  117  127                .ASCII  /ROW/
000456  123  040  101                .ASCII  /S A/
000461  116  104  040                .ASCII  /ND /
000464  103  117  114                .ASCII  /COL/
000467  125  115  116                .ASCII  /UMN/
000472  123  040  104                .ASCII  /S D/
000475  105  124  105                .ASCII  /ETE/
000500  103  124  105                .ASCII  /CTE/
000503  104  040  040                .ASCII  /D  /
000506  127  111  124                .ASCII  /WIT/
000511  110  111  116                .ASCII  /HIN/
000514  040  105  101                .ASCII  / EA/
000517  103  110  040                .ASCII  /CH /
000522  040  105  122                .ASCII  / ER/
000525  122  117  122                .ASCII  /ROR/
000530  111  116  107                .ASCII  /ING/
000533  040  040  103                .ASCII  /  C/
000536  110  111  120                .ASCII  /HIP/
000541  056  000  000                .ASCII  /./<00><00>
000544  045  116  045          P.AAG:  .ASCII  /%N%/
000547  101  011  011                .ASCII  /A/<11><11>
000552  110  117  127                .ASCII  /HOW/
000555  105  126  105                .ASCII  /EVE/
000560  122  040  124                .ASCII  /R T/
000563  110  111  123                .ASCII  /HIS/
000566  040  114  111                .ASCII  / LI/
000571  123  124  040                .ASCII  /ST /
000574  104  117  105                .ASCII  /DOE/
000577  123  040  116                .ASCII  /S N/
000602  117  124  040                .ASCII  /OT /
000605  116  105  103                .ASCII  /NEC/
000610  105  123  123                .ASCII  /ESS/
000613  101  122  111                .ASCII  /ARI/
000616  114  131  040                .ASCII  /LY /
000621  111  116  104                .ASCII  /IND/
000624  111  103  101                .ASCII  /ICA/
000627  124  105  040                .ASCII  /TE /
```

```
000632  124  110  105                .ASCII  /THE/
000635  040  116  125                .ASCII  / NU/
000640  115  102  105                .ASCII  /MBE/
000643  122  040  117                .ASCII  /R O/
000646  106  040  116                .ASCII  /F N/
000651  105  127  114                .ASCII  /EWL/
000654  131  000                     .ASCII  /Y/<00>
000656  045  116  045      P.AAH:    .ASCII  /%N%/
000661  101  011  011                .ASCII  /A/<11><11>
000664  106  101  111                .ASCII  /FAI/
000667  114  111  116                .ASCII  /LIN/
000672  107  040  122                .ASCII  /G R/
000675  117  127  123                .ASCII  /OWS/
000700  040  117  122                .ASCII  / OR/
000703  040  103  117                .ASCII  / CO/
000706  114  125  115                .ASCII  /LUM/
000711  116  123  040                .ASCII  /NS /
000714  102  114  101                .ASCII  /BLA/
000717  123  124  105                .ASCII  /STE/
000722  104  045  116                .ASCII  /D%N/
000725  000                         .ASCII  <00>
000726  111  114  114      P.AAI:    .ASCII  /ILL/
000731  105  107  101                .ASCII  /EGA/
000734  114  040  120                .ASCII  /L P/
000737  122  117  115                .ASCII  /ROM/
000742  040  115  101                .ASCII  / MA/
000745  111  116  124                .ASCII  /INT/
000750  105  116  101                .ASCII  /ENA/
000753  116  103  105                .ASCII  /NCE/
000756  040  120  122                .ASCII  / PR/
000761  117  107  122                .ASCII  /OGR/
000764  101  115  040                .ASCII  /AM /
000767  103  117  115                .ASCII  /COM/
000772  115  101  116                .ASCII  /MAN/
000775  104  040  000                .ASCII  /D /<00>
001000  123  124  101      P.AAJ:    .ASCII  /STA/
001003  122  124  111                .ASCII  /RTI/
001006  116  107  040                .ASCII  /NG /
001011  120  122  117                .ASCII  /PRO/
001014  115  040  115                .ASCII  /M M/
001017  101  111  116                .ASCII  /AIN/
001022  124  105  116                .ASCII  /TEN/
001025  101  116  103                .ASCII  /ANC/
001030  105  000                     .ASCII  /E/<00>
001032  124  131  120      P.AAK:    .ASCII  /TYP/
001035  105  040  123                .ASCII  /E S/
001040  124  101  122                .ASCII  /TAR/
001043  124  040  124                .ASCII  /T T/
001046  117  040  105                .ASCII  /O E/
001051  130  105  103                .ASCII  /XEC/
001054  125  124  105                .ASCII  /UTE/
001057  040  120  122                .ASCII  / PR/
```

```
001062  117  107  122                .ASCII  /OGR/
001065  101  115  000                .ASCII  /AM/<00>
001070  110  101  122      P.AAL:    .ASCII  /HAR/
001073  104  127  101                .ASCII  /DWA/
001076  122  105  040                .ASCII  /RE /
001101  121  125  105                .ASCII  /QUE/
001104  123  124  111                .ASCII  /STI/
001107  117  116  123                .ASCII  /ONS/
001112  040  116  117                .ASCII  / NO/
001115  124  040  101                .ASCII  /T A/
001120  116  123  127                .ASCII  /NSW/
001123  105  122  105                .ASCII  /ERE/
001126  104  040  103                .ASCII  /D C/
001131  117  122  122                .ASCII  /ORR/
001134  105  103  124                .ASCII  /ECT/
001137  114  131  040                .ASCII  /LY /
001142  000  000                     .ASCII  <00><00>
001144  102  114  101      P.AAM:    .ASCII  /BLA/
001147  123  124  111                .ASCII  /STI/
001152  116  107  040                .ASCII  /NG /
001155  123  131  123                .ASCII  /SYS/
001160  124  105  115                .ASCII  /TEM/
001163  054  040  101                .ASCII  /, A/
001166  122  122  101                .ASCII  /RRA/
001171  131  040  117                .ASCII  /Y O/
001174  122  040  102                .ASCII  /R B/
001177  101  116  113                .ASCII  /ANK/
001202  040  116  117                .ASCII  / NO/
001205  124  040  123                .ASCII  /T S/
001210  105  114  105                .ASCII  /ELE/
001213  103  124  105                .ASCII  /CTE/
001216  104  040  000                .ASCII  /D /<00>
001221  000                        .ASCII  <00>
001222  120  122  117      P.AAN:    .ASCII  /PRO/
001225  115  040  115                .ASCII  /M M/
001230  101  111  116                .ASCII  /AIN/
001233  124  101  116                .ASCII  /TAN/
001236  105  116  103                .ASCII  /ENC/
001241  105  040  103                .ASCII  /E C/
001244  117  115  120                .ASCII  /OMP/
001247  114  105  124                .ASCII  /LET/
001252  105  104  040                .ASCII  /ED /
001255  000                        .ASCII  <00>
001256  123  125  120      P.AAO:    .ASCII  /SUP/
001261  120  122  105                .ASCII  /PRE/
001264  123  123  111                .ASCII  /SSI/
001267  116  107  040                .ASCII  /NG /
001272  120  122  117                .ASCII  /PRO/
001275  107  122  101                .ASCII  /GRA/
001300  115  040  105                .ASCII  /M E/
001303  130  105  103                .ASCII  /XEC/
001306  125  124  111                .ASCII  /UTI/
```

```
001311  117  116  040                    .ASCII  /ON /
001314  000  000                         .ASCII  <00><00>
001316  122  105  124          P.AAP:    .ASCII  /RET/
001321  125  122  116                    .ASCII  /URN/
001324  111  116  107                    .ASCII  /ING/
001327  040  124  117                    .ASCII  / TO/
001332  040  104  122                    .ASCII  / DR/
001335  123  076  000                    .ASCII  /S>/<00>
001340  114  125  116          P.AAQ:    .ASCII  /LUN/
001343  040  060  040                    .ASCII  / 0 /
001346  120  055  124                    .ASCII  /P-T/
001351  101  102  114                    .ASCII  /ABL/
001354  105  040  116                    .ASCII  /E N/
001357  117  124  040                    .ASCII  /OT /
001362  120  122  105                    .ASCII  /PRE/
001365  123  105  116                    .ASCII  /SEN/
001370  124  000                         .ASCII  /T/<00>
001372  103  117  116          P.AAR:    .ASCII  /CON/
001375  104  111  124                    .ASCII  /DIT/
001400  111  117  116                    .ASCII  /ION/
001403  040  101  000                    .ASCII  / A/<00>
001406  103  117  116          P.AAS:    .ASCII  /CON/
001411  104  111  124                    .ASCII  /DIT/
001414  111  117  116                    .ASCII  /ION/
001417  040  102  000                    .ASCII  / B/<00>
001422  103  117  116          P.AAT:    .ASCII  /CON/
001425  104  111  124                    .ASCII  /DIT/
001430  111  117  116                    .ASCII  /ION/
001433  040  103  000                    .ASCII  / C/<00>
001436  103  117  116          P.AAU:    .ASCII  /CON/
001441  104  111  124                    .ASCII  /DIT/
001444  111  117  116                    .ASCII  /ION/
00?,47  040  104  040                    .ASCII  / D /
001452  000  000                         .ASCII  <00><00>
001454  125  116  103          P.AAV:    .ASCII  /UNC/
001457  117  116  106                    .ASCII  /ONF/
001462  111  122  115                    .ASCII  /IRM/
001465  105  104  040                    .ASCII  /ED /
001470  106  101  111                    .ASCII  /FAI/
001473  114  111  116                    .ASCII  /LIN/
001476  107  040  103                    .ASCII  /G C/
001501  110  111  120                    .ASCII  /HIP/
001504  000  000                         .ASCII  <00><00>
001506  115  117  122          P.AAW:    .ASCII  /MOR/
001511  105  040  124                    .ASCII  /E T/
001514  110  101  116                    .ASCII  /HAN/
001517  040  117  116                    .ASCII  / ON/
001522  105  040  125                    .ASCII  /E U/
001525  116  111  124                    .ASCII  /NIT/
001530  040  123  105                    .ASCII  / SE/
001533  114  105  103                    .ASCII  /LEC/
001536  124  105  104                    .ASCII  /TED/
```

```
001541  040  106  117                .ASCII  / FO/
001544  122  040  120                .ASCII  /R P/
001547  122  117  115                .ASCII  /ROM/
001552  040  115  101                .ASCII  / MA/
001555  111  116  124                .ASCII  /INT/
001560  000  000                     .ASCII  <00><00>
001562  101  114  114      P.AAX:    .ASCII  /ALL/
001565  040  125  116                .ASCII  / UN/
001570  111  124  123                .ASCII  /ITS/
001573  040  105  130                .ASCII  / EX/
001576  103  105  120                .ASCII  /CEP/
001601  124  040  125                .ASCII  /T U/
001604  116  111  124                .ASCII  /NIT/
001607  040  060  040                .ASCII  / 0 /
001612  127  111  114                .ASCII  /WIL/
001615  114  040  102                .ASCII  /L B/
001620  105  040  111                .ASCII  /E I/
001623  107  116  117                .ASCII  /GNO/
001626  122  105  104                .ASCII  /RED/
001631  000                         .ASCII  <00>
001632  125  116  105      P.AAY:    .ASCII  /UNE/
001635  130  120  105                .ASCII  /XPE/
001640  103  124  105                .ASCII  /CTE/
001643  104  040  123                .ASCII  /D S/
001646  117  106  124                .ASCII  /OFT/
001651  127  101  122                .ASCII  /WAR/
001654  105  040  102                .ASCII  /E B/
001657  125  107  040                .ASCII  /UG /
001662  104  105  124                .ASCII  /DET/
001665  105  103  124                .ASCII  /ECT/
001670  105  104  040                .ASCII  /ED /
001673  116  117  124                .ASCII  /NOT/
001676  111  106  131                .ASCII  /IFY/
001701  040  104  111                .ASCII  / DI/
001704  101  107  040                .ASCII  /AG /
001707  105  116  107                .ASCII  /ENG/
001712  000  000                     .ASCII  <00><00>
001714  120  122  117      P.AAZ:    .ASCII  /PRO/
001717  115  040  115                .ASCII  /M M/
001722  101  111  116                .ASCII  /AIN/
001725  124  101  116                .ASCII  /TAN/
001730  105  116  103                .ASCII  /ENC/
001733  105  040  101                .ASCII  /E A/
001736  102  117  122                .ASCII  /BOR/
001741  124  105  104                .ASCII  /TED/
001744  040  106  117                .ASCII  / FO/
001747  122  040  124                .ASCII  /R T/
001752  110  111  123                .ASCII  /HIS/
001755  040  101  122                .ASCII  / AR/
001760  122  101  131                .ASCII  /RAY/
001763  000                         .ASCII  <00>
001764  116  117  040      P.ABA:    .ASCII  /NO /
```

```
001767  101  104  104                .ASCII  /ADD/
001772  111  124  111                .ASCII  /ITI/
001775  117  116  101                .ASCII  /ONA/
002000  114  040  105                .ASCII  /L E/
002003  122  122  117                .ASCII  /RRO/
002006  122  123  040                .ASCII  /RS /
002011  102  114  101                .ASCII  /BLA/
002014  123  124  105                .ASCII  /STE/
002017  104  040  106                .ASCII  /D F/
002022  117  122  040                .ASCII  /OR /
002025  124  110  111                .ASCII  /THI/
002030  123  040  101                .ASCII  /S A/
002033  122  122  101                .ASCII  /RRA/
002036  131  000                     .ASCII  /Y/<00>
002040  125  116  103          P.ABB: .ASCII  /UNC/
002043  117  122  122                .ASCII  /ORR/
002046  105  103  124                .ASCII  /ECT/
002051  101  102  114                .ASCII  /ABL/
002054  105  040  105                .ASCII  /E E/
002057  122  122  117                .ASCII  /RRO/
002062  122  123  040                .ASCII  /RS /
002065  104  105  124                .ASCII  /DET/
002070  105  103  124                .ASCII  /ECT/
002073  105  104  040                .ASCII  /ED /
002076  101  106  124                .ASCII  /AFT/
002101  105  122  040                .ASCII  /ER /
002104  102  114  101                .ASCII  /BLA/
002107  123  124  111                .ASCII  /STI/
002112  116  107  000                .ASCII  /NG/<00>
002115  000                        .ASCII  <00>
002116  104  101  124          P.ABC: .ASCII  /DAT/
002121  101  040  103                .ASCII  /A C/
002124  114  117  103                .ASCII  /LOC/
002127  113  040  102                .ASCII  /K B/
002132  111  124  040                .ASCII  /IT /
002135  106  101  111                .ASCII  /FAI/
002140  114  105  104                .ASCII  /LED/
002143  040  124  117                .ASCII  / TO/
002146  040  122  105                .ASCII  / RE/
002151  123  105  124                .ASCII  /SET/
002154  040  101  106                .ASCII  / AF/
002157  124  105  122                .ASCII  /TER/
002162  040  120  122                .ASCII  / PR/
002165  117  115  040                .ASCII  /OM /
002170  127  122  111                .ASCII  /WRI/
002173  124  105  000                .ASCII  /TE/<00>
002176  122  105  120          P.ABD: .ASCII  /REP/
002201  114  101  103                .ASCII  /LAC/
002204  105  040  101                .ASCII  /E A/
002207  122  122  101                .ASCII  /RRA/
002212  131  040  104                .ASCII  /Y D/
002215  101  124  101                .ASCII  /ATA/
```

```
002220  040  115  117                    .ASCII  / MO/
002223  104  125  114                    .ASCII  /DUL/
002226  105  040  115                    .ASCII  /E M/
002231  067  063  066                    .ASCII  /736/
002234  063  000                         .ASCII  /3/<00>
002236  115  105  115          P.ABE:    .ASCII  /MEM/
002241  117  122  131                    .ASCII  /ORY/
002244  040  105  122                    .ASCII  / ER/
002247  122  117  122                    .ASCII  /ROR/
002252  123  040  123                    .ASCII  /S S/
002255  124  111  114                    .ASCII  /TIL/
002260  114  040  105                    .ASCII  /L E/
002263  130  111  123                    .ASCII  /XIS/
002266  124  040  101                    .ASCII  /T A/
002271  106  124  105                    .ASCII  /FTE/
002274  122  040  102                    .ASCII  /R B/
002277  114  101  123                    .ASCII  /LAS/
002302  124  111  116                    .ASCII  /TIN/
002305  107  000  000                    .ASCII  /G/<00><00>
002310  125  116  105          P.ABF:    .ASCII  /UNE/
002313  130  120  105                    .ASCII  /XPE/
002316  103  124  105                    .ASCII  /CTE/
002321  104  040  104                    .ASCII  /D D/
002324  122  111  126                    .ASCII  /RIV/
002327  105  040  105                    .ASCII  /E E/
002332  122  122  117                    .ASCII  /RRO/
002335  122  123  040                    .ASCII  /RS /
002340  104  105  124                    .ASCII  /DET/
002343  105  103  124                    .ASCII  /ECT/
002346  105  104  000                    .ASCII  /ED/<00>
002351  000                            .ASCII  <00>
002352  117  103  103          P.ABG:    .ASCII  /OCC/
002355  125  122  105                    .ASCII  /URE/
002360  104  040  104                    .ASCII  /D D/
002363  125  122  111                    .ASCII  /URI/
002366  116  107  040                    .ASCII  /NG /
002371  101  040  115                    .ASCII  /A M/
002374  101  123  123                    .ASCII  /ASS/
002377  040  102  125                    .ASCII  / BU/
002402  123  040  122                    .ASCII  /S R/
002405  105  101  104                    .ASCII  /EAD/
002410  040  124  122                    .ASCII  / TR/
002413  101  116  123                    .ASCII  /ANS/
002416  106  105  122                    .ASCII  /FER/
002421  000                            .ASCII  <00>
002422  117  103  103          P.ABH:    .ASCII  /OCC/
002425  125  122  105                    .ASCII  /URE/
002430  104  040  104                    .ASCII  /D D/
002433  125  122  111                    .ASCII  /URI/
002436  116  107  040                    .ASCII  /NG /
002441  101  040  115                    .ASCII  /A M/
002444  101  123  123                    .ASCII  /ASS/
```

```
002447  040  102  125            .ASCII  / BU/
002452  123  040  127            .ASCII  /S W/
002455  122  111  124            .ASCII  /RIT/
002460  105  040  124            .ASCII  /E T/
002463  122  101  116            .ASCII  /RAN/
002466  123  106  105            .ASCII  /SFE/
002471  122  000  000            .ASCII  /R/<00><00>
002474  117  103  103  P.ABI:    .ASCII  /OCC/
C02477  125  122  105            .ASCII  /URE/
002502  104  040  104            .ASCII  /D D/
002505  125  122  111            .ASCII  /URI/
002510  116  107  040            .ASCII  /NG /
002513  101  040  115            .ASCII  /A M/
002516  101  123  123            .ASCII  /ASS/
002521  040  102  125            .ASCII  / BU/
002524  123  040  127            .ASCII  /S W/
002527  122  111  124            .ASCII  /RIT/
002532  105  040  103            .ASCII  /E C/
002535  110  105  103            .ASCII  /HEC/
002540  113  040  124            .ASCII  /K T/
002543  122  101  116            .ASCII  /RAN/
002546  123  106  105            .ASCII  /SFc/
002551  122  000  000            .ASCII  /R/<00><00>
002554  000  000       P.ABJ:    .ASCII  <00><00>
002556  045  124  045  P.ABK:    .ASCII  /%T%/
002561  116  000  000            .ASCII  /N/<00><00>
002564  045  124  045  P.ABL:    .ASCII  /%T%/
002567  124  045  116            .ASCII  /T%N/
002572  000  000                 .ASCII  <00><00>
002574  045  124  045  P.ABM:    .ASCII  /%T%/
002577  124  045  124            .ASCII  /T%T/
002602  045  116  000            .ASCII  /%N/<00>
002605  000                      .ASCII  <00>
002606  045  124  045  P.ABN:    .ASCII  /%T%/
002611  124  045  124            .ASCII  /T%T/
002614  045  124  045            .ASCII  /%T%/
002617  116  000  000            .ASCII  /N/<00><00>
002622  045  116  000  P.ABO:    .ASCII  /%N/<00>
002625  000                      .ASCII  <00>
002626  045  116  000  P.ABP:    .ASCII  /%N/<00>
002631  000                      .ASCII  <00>
002632  045  116  045  P.ABQ:    .ASCII  /%N%/
002635  116  000  000            .ASCII  /N/<00><00>
002640  045  124  045  P.ABR:    .ASCII  /%T%/
002643  117  061  045            .ASCII  /01%/
002646  116  000                 .ASCII  /N/<00>
002650  045  124  045  P.ABS:    .ASCII  /%T%/
002653  117  066  045            .ASCII  /06%/
002656  116  000                 .ASCII  /N/<00>
002660  045  124  045  P.ABT:    .ASCII  /%T%/
002663  104  062  045            .ASCII  /D2%/
002666  116  000                 .ASCII  /N/<00>
```

```
002670  045  101  104        P.ABU:  .ASCII  /%AD/
002673  122  111  126                .ASCII  /RIV/
002676  105  040  045                .ASCII  /E %/
002701  117  061  045                .ASCII  /O1%/
002704  101  040  123                .ASCII  /A S/
002707  105  114  105                .ASCII  /ELE/
002712  103  124  105                .ASCII  /CTE/
002715  104  045  116                .ASCII  /D%N/
002720  000  000                     .ASCII  <00><00>
002722  045  123  045        P.ABV:  .ASCII  /%S%/
002725  123  045  123                .ASCII  /S%S/
002730  045  123  045                .ASCII  /%S%/
002733  123  045  123                .ASCII  /S%S/
002736  045  123  045                .ASCII  /%S%/
002741  123  045  101                .ASCII  /S%A/
002744  124  105  123                .ASCII  /TES/
002747  124  111  116                .ASCII  /TIN/
002752  107  040  101                .ASCII  /G A/
002755  122  122  101                .ASCII  /RRA/
002760  131  040  045                .ASCII  /Y %/
002763  104  062  045                .ASCII  /D2%/
002766  116  000                     .ASCII  /N/<00>
002770  045  123  045        P.ABW:  .ASCII  /%S%/
002773  123  045  123                .ASCII  /S%S/
002776  045  123  045                .ASCII  /%S%/
003001  123  045  123                .ASCII  /S%S/
003004  045  123  045                .ASCII  /%S%/
003007  123  045  123                .ASCII  /S%S/
003012  045  123  045                .ASCII  /%S%/
003015  123  045  123                .ASCII  /S%S/
003020  045  123  045                .ASCII  /%S%/
003023  123  045  123                .ASCII  /S%S/
003026  045  123  045                .ASCII  /%S%/
003031  101  102  101                .ASCII  /ABA/
003034  116  113  045                .ASCII  /NK%/
003037  123  045  123                .ASCII  /S%S/
003042  045  123  045                .ASCII  /%S%/
003045  104  061  045                .ASCII  /D1%/
003050  116  000                     .ASCII  /N/<00>
003052  045  101  101        P.ABX:  .ASCII  /%AA/
003055  122  122  101                .ASCII  /RRA/
003060  131  045  123                .ASCII  /Y%S/
003063  045  104  062                .ASCII  /%D2/
003066  045  123  045                .ASCII  /%S%/
003071  101  102  101                .ASCII  /ABA/
003074  116  113  045                .ASCII  /NK%/
003077  123  045  117                .ASCII  /S%O/
003102  061  045  123                .ASCII  /1%S/
003105  045  101  102                .ASCII  /%AB/
003110  111  124  040                .ASCII  /IT /
003113  043  045  123                .ASCII  /#%S/
003116  045  104  062                .ASCII  /%D2/
```

```
003121  045  116  000          .ASCII  /%N/<00>
003124  045  101  101  P.ABY:   .ASCII  /%AA/
003127  122  122  101          .ASCII  /RRA/
003132  131  045  123          .ASCII  /Y%S/
003135  045  104  062          .ASCII  /%D2/
003140  045  123  045          .ASCII  /%S%/
003143  101  102  101          .ASCII  /ABA/
003146  116  113  045          .ASCII  /NK%/
003151  123  045  117          .ASCII  /S%O/
003154  061  045  123          .ASCII  /1%S/
003157  045  101  116          .ASCII  /%AN/
003162  111  102  102          .ASCII  /IBB/
003165  114  105  045          .ASCII  /LE%/
003170  123  045  104          .ASCII  /S%D/
003173  062  045  116          .ASCII  /2%N/
003176  000  000               .ASCII  <00><00>
003200  045  101  122  P.ABZ:   .ASCII  /%AR/
003203  105  120  114          .ASCII  /EPL/
003206  101  103  105          .ASCII  /ACE/
003211  040  101  122          .ASCII  / AR/
003214  122  101  131          .ASCII  /RAY/
003217  040  115  117          .ASCII  / MO/
003222  104  125  114          .ASCII  /DUL/
003225  105  045  123          .ASCII  /E%S/
003230  045  104  062          .ASCII  /%D2/
003233  045  123  045          .ASCII  /%S%/
003236  123  045  101          .ASCII  /S%A/
003241  117  122  040          .ASCII  /OR /
003244  122  105  122          .ASCII  /RER/
003247  125  116  040          .ASCII  /UN /
003252  120  122  117          .ASCII  /PRO/
003255  107  122  101          .ASCII  /GRA/
003260  115  045  116          .ASCII  /M%N/
003263  000                    .ASCII  <00>
003264  045  101  101  P.ACA:   .ASCII  /%AA/
003267  122  122  101          .ASCII  /RRA/
003272  131  045  123          .ASCII  /Y%S/
003275  045  104  062          .ASCII  /%D2/
003300  045  123  045          .ASCII  /%S%/
003303  101  102  101          .ASCII  /ABA/
003306  116  113  045          .ASCII  /NK%/
003311  123  045  117          .ASCII  /S%O/
003314  061  045  116          .ASCII  /1%N/
003317  000                    .ASCII  <00>
003320  045  123  045  P.ACB:   .ASCII  /%S%/
003323  123  045  123          .ASCII  /S%S/
003326  045  123  045          .ASCII  /%S%/
003331  101  115  114          .ASCII  /AML/
003334  103  123  061          .ASCII  /CS1/
003337  045  123  045          .ASCII  /%S%/
003342  123  045  123          .ASCII  /S%S/
003345  045  123  045          .ASCII  /%S%/
```

```
003350      123     045     123                 .ASCII  /S%S/
003353      045     123     045                 .ASCII  /%S%/
003356      123     045     123                 .ASCII  /S%S/
003361      045     123     045                 .ASCII  /%S%/
003364      123     045     123                 .ASCII  /S%S/
003367      045     123     045                 .ASCII  /%S%/
003372      123     045     123                 .ASCII  /S%S/
003375      045     123     045                 .ASCII  /%S%/
003400      117     066     045                 .ASCII  /06%/
003403      116     000     000                 .ASCII  /N/<00><00>
003406      045     123     045         P.ACC:  .ASCII  /%S%/
003411      123     045     123                 .ASCII  /S%S/
003414      045     123     045                 .ASCII  /%S%/
003417      101     115     114                 .ASCII  /AML/
003422      127     103     045                 .ASCII  /WC%/
003425      123     045     123                 .ASCII  /S%S/
003430      045     123     045                 .ASCII  /%S%/
003433      123     045     123                 .ASCII  /S%S/
003436      045     123     045                 .ASCII  /%S%/
003441      123     045     123                 .ASCII  /S%S/
003444      045     123     045                 .ASCII  /%S%/
003447      123     045     123                 .ASCII  /S%S/
003452      045     123     045                 .ASCII  /%S%/
003455      123     045     123                 .ASCII  /S%S/
003460      045     123     045                 .ASCII  /%S%/
003463      123     045     117                 .ASCII  /S%O/
003466      066     045     116                 .ASCII  /6%N/
003471      000                                 .ASCII  <00>
003472      045     123     045         P.ACD:  .ASCII  /%S%/
003475      123     045     123                 .ASCII  /S%S/
003500      045     123     045                 .ASCII  /%S%/
003503      101     115     114                 .ASCII  /AML/
003506      102     101     045                 .ASCII  /BA%/
003511      123     045     123                 .ASCII  /S%S/
003514      045     123     045                 .ASCII  /%S%/
003517      123     045     123                 .ASCII  /S%S/
003522      045     123     045                 .ASCII  /%S%/
003525      123     045     123                 .ASCII  /S%S/
003530      045     123     045                 .ASCII  /%S%/
003533      123     045     123                 .ASCII  /S%S/
003536      045     123     045                 .ASCII  /%S%/
003541      123     045     123                 .ASCII  /S%S/
003544      045     123     045                 .ASCII  /%S%/
003547      123     045     117                 .ASCII  /S%O/
003552      066     045     116                 .ASCII  /6%N/
003555      000                                 .ASCII  <00>
003556      045     123     045         P.ACE:  .ASCII  /%S%/
003561      123     045     123                 .ASCII  /S%S/
003564      045     123     045                 .ASCII  /%S%/
003567      101     115     114                 .ASCII  /AML/
003572      104     101     045                 .ASCII  /DA%/
003575      123     045     123                 .ASCII  /S%S/
```

```
003600  045  123  045            .ASCII  /%S%/
003603  123  045  123            .ASCII  /S%S/
003606  045  123  045            .ASCII  /%S%/
003611  123  045  123            .ASCII  /S%S/
003614  045  123  045            .ASCII  /%S%/
003617  123  045  123            .ASCII  /S%S/
003622  045  123  045            .ASCII  /%S%/
003625  123  045  123            .ASCII  /S%S/
003630  045  123  045            .ASCII  /%S%/
003633  123  045  117            .ASCII  /S%O/
003636  066  045  116            .ASCII  /6%N/
003641  000                      .ASCII  <00>
003642  045  123  045  P.ACF:    .ASCII  /%S%/
003645  123  045  123            .ASCII  /S%S/
003650  045  123  045            .ASCII  /%S%/
003653  101  115  114            .ASCII  /AML/
003656  103  123  062            .ASCII  /CS2/
003661  045  123  045            .ASCII  /%S%/
003664  123  045  123            .ASCII  /S%S/
003667  045  123  045            .ASCII  /%S%/
003672  123  045  123            .ASCII  /S%S/
003675  045  123  045            .ASCII  /%S%/
003700  123  045  123            .ASCII  /S%S/
003703  045  123  045            .ASCII  /%S%/
003706  123  045  123            .ASCII  /S%S/
003711  045  123  045            .ASCII  /%S%/
003714  123  045  123            .ASCII  /S%S/
003717  045  123  045            .ASCII  /%S%/
003722  117  066  045            .ASCII  /O6%/
003725  116  000  000            .ASCII  /N/<00><00>
003730  045  123  045  P.ACG:    .ASCII  /%S%/
003733  123  045  123            .ASCII  /S%S/
003736  045  123  045            .ASCII  /%S%/
003741  101  115  114            .ASCII  /AML/
003744  104  123  045            .ASCII  /DS%/
003747  123  045  123            .ASCII  /S%S/
003752  045  123  045            .ASCII  /%S%/
003755  123  045  123            .ASCII  /S%S/
003760  045  123  045            .ASCII  /%S%/
003763  123  045  123            .ASCII  /S%S/
003766  045  123  045            .ASCII  /%S%/
003771  123  045  123            .ASCII  /S%S/
003774  045  123  045            .ASCII  /%S%/
003777  123  045  123            .ASCII  /S%S/
004002  045  123  045            .ASCII  /%S%/
004005  123  045  117            .ASCII  /S%O/
004010  066  045  116            .ASCII  /6%N/
004013  000                      .ASCII  <00>
004014  045  123  045  P.ACH:    .ASCII  /%S%/
004017  123  045  123            .ASCII  /S%S/
004022  045  123  045            .ASCII  /%S%/
004025  101  115  114            .ASCII  /AML/
```

```
004030  105  122  045              .ASCII  /ER%/
004033  123  045  123              .ASCII  /S%S/
004036  045  123  045              .ASCII  /%S%/
004041  123  045  123              .ASCII  /S%S/
004044  045  123  045              .ASCII  /%S%/
004047  123  045  123              .ASCII  /S%S/
004052  045  123  045              .ASCII  /%S%/
004055  123  045  123              .ASCII  /S%S/
004060  045  123  045              .ASCII  /%S%/
004063  123  045  123              .ASCII  /S%S/
004066  045  123  045              .ASCII  /%S%/
004071  123  045  117              .ASCII  /S%O/
004074  066  045  116              .ASCII  /6%N/
004077  000                       .ASCII  <00>
004100  045  123  045    P.ACI:    .ASCII  /%S%/
004103  123  045  123              .ASCII  /S%S/
004106  045  123  045              .ASCII  /%S%/
004111  101  115  114              .ASCII  /AML/
004114  101  123  045              .ASCII  /AS%/
004117  123  045  123              .ASCII  /S%S/
004122  045  123  045              .ASCII  /%S%/
004125  123  045  123              .ASCII  /S%S/
004130  045  123  045              .ASCII  /%S%/
004133  123  045  123              .ASCII  /S%S/
004136  045  123  045              .ASCII  /%S%/
004141  123  045  123              .ASCII  /S%S/
004144  045  123  045              .ASCII  /%S%/
004147  123  045  123              .ASCII  /S%S/
004152  045  123  045              .ASCII  /%S%/
004155  123  045  117              .ASCII  /S%O/
004160  066  045  116              .ASCII  /6%N/
004163  000                       .ASCII  <00>
004164  045  123  045    P.ACJ:    .ASCII  /%S%/
004167  123  045  123              .ASCII  /S%S/
004172  045  123  045              .ASCII  /%S%/
004175  101  115  114              .ASCII  /AML/
004200  115  122  045              .ASCII  /MR%/
004203  123  045  123              .ASCII  /S%S/
004206  045  123  045              .ASCII  /%S%/
004211  123  045  123              .ASCII  /S%S/
004214  045  123  045              .ASCII  /%S%/
004217  123  045  123              .ASCII  /S%S/
004222  045  123  045              .ASCII  /%S%/
004225  123  045  123              .ASCII  /S%S/
004230  045  123  045              .ASCII  /%S%/
004233  123  045  123              .ASCII  /S%S/
004236  045  123  045              .ASCII  /%S%/
004241  123  045  117              .ASCII  /S%O/
004244  066  045  116              .ASCII  /6%N/
004247  000                       .ASCII  <00>
004250  045  123  045    P.ACK:    .ASCII  /%S%/
004253  123  045  123              .ASCII  /S%S/
```

```
004256  045  123  045                .ASCII  /%S%/
004261  101  115  114                .ASCII  /AML/
004264  104  124  045                .ASCII  /DT%/
004267  123  045  123                .ASCII  /S%S/
004272  045  123  045                .ASCII  /%S%/
004275  123  045  123                .ASCII  /S%S/
004300  045  123  045                .ASCII  /%S%/
004303  123  045  123                .ASCII  /S%S/
004306  045  123  045                .ASCII  /%S%/
004311  123  045  123                .ASCII  /S%S/
004314  045  123  045                .ASCII  /%S%/
004317  123  045  123                .ASCII  /S%S/
004322  045  123  045                .ASCII  /%S%/
004325  123  045  117                .ASCII  /S%O/
004330  066  045  116                .ASCII  /6%N/
004333  000                         .ASCII  <00>
004334  045  123  045     P.ACL:     .ASCII  /%S%/
004337  123  045  123                .ASCII  /S%S/
004342  045  123  045                .ASCII  /%S%/
004345  101  115  114                .ASCII  /AML/
004350  123  116  045                .ASCII  /SN%/
004353  123  045  123                .ASCII  /S%S/
004356  045  123  045                .ASCII  /%S%/
004361  123  045  123                .ASCII  /S%S/
004364  045  123  045                .ASCII  /%S%/
004367  123  045  123                .ASCII  /S%S/
004372  045  123  045                .ASCII  /%S%/
004375  123  045  123                .ASCII  /S%S/
004400  045  123  045                .ASCII  /%S%/
004403  123  045  123                .ASCII  /S%S/
004406  045  123  045                .ASCII  /%S%/
004411  123  045  117                .ASCII  /S%O/
004414  066  045  116                .ASCII  /6%N/
004417  000                         .ASCII  <00>
004420  045  123  045     P.ACM:     .ASCII  /%S%/
004423  123  045  123                .ASCII  /S%S/
004426  045  123  045                .ASCII  /%S%/
004431  101  115  114                .ASCII  /AML/
004434  105  105  045                .ASCII  /EE%/
004437  123  045  123                .ASCII  /S%S/
004442  045  123  045                .ASCII  /%S%/
004445  123  045  123                .ASCII  /S%S/
004450  045  123  045                .ASCII  /%S%/
004453  123  045  123                .ASCII  /S%S/
004456  045  123  045                .ASCII  /%S%/
004461  123  045  123                .ASCII  /S%S/
004464  045  123  045                .ASCII  /%S%/
004467  123  045  123                .ASCII  /S%S/
004472  045  123  045                .ASCII  /%S%/
004475  123  045  117                .ASCII  /S%O/
004500  066  045  116                .ASCII  /6%N/
004503  000                         .ASCII  <00>
```

```
004504  045  123  045        P.ACN:  .ASCII  /%S%/
004507  123  045  123                .ASCII  /S%S/
004512  045  123  045                .ASCII  /%S%/
004515  101  115  114                .ASCII  /AML/
004520  105  114  045                .ASCII  /EL%/
004523  123  045  123                .ASCII  /S%S/
004526  045  123  045                .ASCII  /%S%/
004531  123  045  123                .ASCII  /S%S/
004534  045  123  045                .ASCII  /%S%/
004537  123  045  123                .ASCII  /S%S/
004542  045  123  045                .ASCII  /%S%/
004545  123  045  123                .ASCII  /S%S/
004550  045  123  045                .ASCII  /%S%/
004553  123  045  123                .ASCII  /S%S/
004556  045  123  045                .ASCII  /%S%/
004561  123  045  117                .ASCII  /S%O/
004564  066  045  116                .ASCII  /6%N/
004567  000                        .ASCII  <00>
004570  045  123  045        P.ACO:  .ASCII  /%S%/
004573  123  045  123                .ASCII  /S%S/
004576  045  123  045                .ASCII  /%S%/
004601  123  045  123                .ASCII  /S%S/
004604  045  123  045                .ASCII  /%S%/
004607  123  045  123                .ASCII  /S%S/
004612  045  123  045                .ASCII  /%S%/
004615  101  122  105                .ASCII  /ARE/
004620  107  111  123                .ASCII  /GIS/
004623  124  105  122                .ASCII  /TER/
004626  040  040  104                .ASCII  /  D/
004631  125  115  120                .ASCII  /UMP/
004634  045  116  045                .ASCII  /%N%/
004637  116  000  000                .ASCII  /N/<00><00>
004642  045  123  045        P.ACP:  .ASCII  /%S%/
004645  123  045  101                .ASCII  /S%A/
004650  122  105  107                .ASCII  /REG/
004653  111  123  124                .ASCII  /IST/
004656  105  122  040                .ASCII  /ER /
004661  116  101  115                .ASCII  /NAM/
004664  105  045  123                .ASCII  /E%S/
004667  045  123  045                .ASCII  /%S%/
004672  123  045  123                .ASCII  /S%S/
004675  045  123  045                .ASCII  /%S%/
004700  123  045  101                .ASCII  /S%A/
004703  103  117  116                .ASCII  /CON/
004706  124  105  116                .ASCII  /TEN/
004711  124  123  045                .ASCII  /TS%/
004714  116  000                    .ASCII  /N/<00>
004716  000  000            P.ACQ:  .ASCII  <00><00>


000000                              .PSECT  $OWN$,  D
```

```
000000                          PM.COUNT:
                                        .BLKW   1
000002                          PM.LOG: .BLKW   400
001002                          COL.CNT.TBL:
                                        .BLKW   200
001402                          REM.TBL:.BLKW   144
001712                          COPIED.REM.TBL:
                                        .BLKW   144
002222                          COL.PTR:.BLKW   5
002234                          WRT.BUF:.BLKW   2376
007230                          RD.BUF: .BLKW   400
010230                          ERROR.MAP:
                                        .BLKW   10000
030230                          CHIP.TBL:
                                        .BLKW   47
030346                          TMP.BLST.TBL:
                                        .BLKW   12
030372                          BLAST.TBL:
                                        .BLKW   4000
040372                          COL.BASE:
                                        .BLKW   1
040374                          ML.ADDR:.BLKW   1
040376                          MAX.CHIP.COL:
                                        .BLKW   1
040400                          ML.DUT: .BLKW   1
040402                          ARR$BNK.SEL:
                                        .BLKW   1
040404                          TSTED.BNK:
                                        .BLKW   1
040406                          INC.BNK:.BLKW   1
040410                          INC.ARR:.BLKW   1
040412                          ARR.SEL.POS:
                                        .BLKW   1
040414                          BNK.SEL.POS:
                                        .BLKW   1
040416                          BNK.NUM.SEC:
                                        .BLKW   1
040420                          BAD.BNK.REG:
                                        .BLKB   1
                                        .EVEN
040422                          FLG.REG:.BLKW   1
040424                          DEGRADE.MOD.REG:
                                        .BLKB   1
                                        .EVEN
040426                          SIZE:   .BLKW   1
040430                          DST:    .BLKW   1
040432                          SRC:    .BLKW   1
040434                          LST.TSTED.ARR:
                                        .BLKW   1
040436                          RAND.PASS:
                                        .BLKW   1
```

```
                                .GLOBL  RANGEN, SEED1, SEED2, SEED3, RANDAT
                                .GLOBL  L$UNIT

100000                  BIT15==         -100000
040000                  BIT14==         40000
020000                  BIT13==         20000
010000                  BIT12==         10000
004000                  BIT11==         4000
002000                  BIT10==         2000
001000                  BIT09==         1000
000400                  BIT08==         400
000200                  BIT07==         200
000100                  BIT06==         100
000040                  BIT05==         40
000020                  BIT04==         20
000010                  BIT03==         10
000004                  BIT02==         4
000002                  BIT01==         2
000001                  BIT00==         1
001000                  BIT9==          1000
000400                  BIT8==          400
000200                  BIT7==          200
000100                  BIT6==          100
000040                  BIT5==          40
000020                  BIT4==          20
000010                  BIT3==          10
000004                  BIT2==          4
000002                  BIT1==          2
000001                  BIT0==          1
000040                  EF.START==      40
000037                  EF.RESTART==    37
000036                  EF.CONTINUE==   36
000035                  EF.NEW==        35
000034                  EF.PWR==        34
000340                  PRI07==         340
000300                  PRI06==         300
000240                  PRI05==         240
000200                  PRI04==         200
000140                  PRI03==         140
000100                  PRI02==         100
000040                  PRI01==         40
000000                  PRI00==         0
000004                  EVL==           4
000010                  LOT==           10
000020                  ADR==           20
000040                  IDU==           40
000100                  ISR==           100
000200                  UAM==           200
000400                  BOE==           400
001000                  PNT==           1000
```

```
002000                          PRI==           2000
004000                          IXE==           4000
010000                          IBE==           10000
020000                          IER==           20000
040000                          LOE==           40000
100000                          HOE==           -100000
000000'                         FMT16=          P.AAA
000104'                         PM$HEADER=      P.AAB
000170'                         PM.HEADER=      P.AAC
000244'                         PM.14.MSG=      P.AAD
000324'                         PM.10.MSG=      P.AAE
000434'                         PM.11.MSG=      P.AAF
000544'                         PM.12.MSG=      P.AAG
000656'                         PM.13.MSG=      P.AAH
000726'                         ILL.CMD.MSG=    P.AAI
001000'                         BGN.MSG=        P.AAJ
001032'                         START.MSG=      P.AAK
001070'                         HQ.ERR.MSG=     P.AAL
001144'                         HQ.MSG=         P.AAM
001222'                         END.MSG=        P.AAN
001256'                         SUPRES.MSG=     P.AAO
001316'                         RET.DRS.MSG=    P.AAP
001340'                         LUN.MISS.MSG=   P.AAQ
001372'                         CON.A.MSG=      P.AAR
001406'                         CON.B.MSG=      P.AAS
001422'                         CON.C.MSG=      P.AAT
001436'                         CON.D.MSG=      P.AAU
001454'                         UNC.CHIP.MSG=   P.AAV
001506'                         GTR.MSG=        P.AAW
001562'                         UNIT.SEL.MSG=   P.AAX
001632'                         SW.BUG.MSG=     P.AAY
001714'                         ABORT.MSG=      P.AAZ
001764'                         NO.AD.MSG=      P.ABA
002040'                         UNC.ERR.MSG=    P.ABB
002116'                         TIME.OUT.MSG=   P.ABC
002176'                         REP.M7363.MSG=  P.ABD
002236'                         MEM.ERR.MSG=    P.ABE
002310'                         UNX.DRV.ERR.MSG= P.ABF
002352'                         RD.XFER.MSG=    P.ABG
002422'                         WRT.XFER.MSG=   P.ABH
002474'                         WT.CHK.XFER.MSG= P.ABI
002554'                         X.MSG=          P.ABJ
002556'                         ONE.MSG=        P.ABK
002564'                         TWO.MSG=        P.ABL
002574'                         THREE.MSG=      P.ABM
002606'                         FOUR.MSG=       P.ABN
002622'                         CRLF=           P.ABO
002626'                         LF=             P.ABP
002632'                         LFS=            P.ABQ
002640'                         0.1.PRINT=      P.ABR
002650'                         0.6.PRINT=      P.ABS
002660'                         D.2.PRINT=      P.ABT
```

```
        002670'                         DRV.SEL.PRINT=      P.ABU
        002722'                         ARR.SEL.PRINT=      P.ABV
        002770'                         BNK.SEL.PRINT=      P.ABW
        003052'                         A.B.C.PRINT=        P.ABX
        003124'                         A.B.N.PRINT=        P.ABY
        003200'                         REP.PRINT=          P.ABZ
        003264'                         A.B.PRINT=          P.ACA
        003320'                         CS1.PRINT=          P.ACB
        003406'                         WC.PRINT=           P.ACC
        003472'                         BA.PRINT=           P.ACD
        003556'                         DA.PRINT=           P.ACE
        003642'                         CS2.PRINT=          P.ACF
        003730'                         DS.PRINT=           P.ACG
        004014'                         ER.PRINT=           P.ACH
        004100'                         AS.PRINT=           P.ACI
        004164'                         MR.PRINT=           P.ACJ
        004250'                         DT.PRINT=           P.ACK
        004334'                         SN.PRINT=           P.ACL
        004420'                         EE.PRINT=           P.ACM
        004504'                         EL.PRINT=           P.ACN
        004570'                         HEADER.PRINT=       P.ACO
        004642'                         COL.DESC.PRINT=     P.ACP
        004716'                         X.PRINT=            P.ACQ


                                        .SBTTL  DUMPER EXTENDED MESSAGE PRINTING SECTION
000000                                  .PSECT  $CODE$

000000  004767  000000V               DUMPER::JSR     PC,M$DUMPER                     ;       2206
000004  104423                                TRAP    23
000006  000207                                RTS     PC

; Routine Size:  4 words
; Maximum stack depth per invocation:  0 words

;       2207  PRINTB (HEADER_PRINT);                                    !Print the header
;       2208  PRINTB (COL_DESC_PRINT);                                  !Describe the column headings
;       2209  PRINTB (CS1_PRINT, .ML_ADDR [MLCS1, ML_ALL]);             !Print mlcs1 contents
;       2210  PRINTB (WC_PRINT, .ML_ADDR [MLWC, ML_ALL]);               !Print mlwc contents
;       2211  PRINTB (BA_PRINT, .ML_ADDR [MLBA, ML_ALL]);               !Print mlba contents
;       2212  PRINTB (DA_PRINT, .ML_ADDR [MLDA, ML_ALL]);               !Print mlda contents
;       2213  PRINTB (CS2_PRINT, .ML_ADDR [MLCS2, ML_ALL]);             !Print mlcs2 contents
;       2214  PRINTB (DS_PRINT, .ML_ADDR [MLDS, ML_ALL]);               !Print mlds contents
;       2215  PRINTB (ER_PRINT, .ML_ADDR [MLER, ML_ALL]);               !Print mler contents
;       2216  PRINTB (AS_PRINT, .ML_ADDR [MLAS, ML_ALL]);               !Print mlas contents
;       2217  PRINTB (MR_PRINT, .ML_ADDR [MLMR, ML_ALL]);               !Print mlmr contents
;       2218  PRINTB (DT_PRINT, .ML_ADDR [MLDT, ML_ALL]);               !Print mldt contents
;       2219  PRINTB (SN_PRINT, .ML_ADDR [MLSN, ML_ALL]);               !Print mlsn contents
;       2220  PRINTB (EE_PRINT, .ML_ADDR [MLEE, ML_ALL]);               !Print mlee contents
;       2221  PRINTB (EL_PRINT, .ML_ADDR [MLEL, ML_ALL]);               !Print mlel contents
;       2222  ENDMSG;
```

```
                                        .SBTTL  M$DUMPER EXTENDED MESSAGE PRINTING SECTION
000010  162706  000032             M$DUMPER:
                                        SUB     #32,SP                          ;                               2206
000014  012746  004570'                 MOV     #HEADER.PRINT,-(SP)             ;                               2207
000020  012746  000001                  MOV     #1,-(SP)
000024  010600                          MOV     SP,R0                           ; SP,*
000026  104414                          TRAP    14
000030  012716  004642'                 MOV     #COL.DESC.PRINT,(SP)            ;                               2208
000034  012746  000001                  MOV     #1,-(SP)
000040  010600                          MOV     SP,R0                           ; SP,*
000042  104414                          TRAP    14
000044  017766  040374'  000036         MOV     @ML.ADDR,36(SP)                 ; *,ML.REG                      2209
000052  016616  000036                  MOV     36(SP),(SP)                     ; ML.REG,*
000056  012746  003320'                 MOV     #CS1.PRINT,-(SP)
000062  012746  000002                  MOV     #2,-(SP)
000066  010600                          MOV     SP,R0                           ; SP,*
000070  104414                          TRAP    14
000072  016700  040374'                 MOV     ML.ADDR,R0                      ;                               2210
000076  016066  000002  000040         MOV     2(R0),40(SP)                    ; *,ML.REG
000104  016616  000040                  MOV     40(SP),(SP)                     ; ML.REG,*
000110  012746  003406'                 MOV     #WC.PRINT,-(SP)
000114  012746  000002                  MOV     #2,-(SP)
000120  010600                          MOV     SP,R0                           ; SP,*
000122  104414                          TRAP    14
000124  016700  040374'                 MOV     ML.ADDR,R0                      ;                               2211
000130  016066  000004  000042         MOV     4(R0),42(SP)                    ; *,ML.REG
000136  016616  000042                  MOV     42(SP),(SP)                     ; ML.REG,*
000142  012746  003472'                 MOV     #BA.PRINT,-(SP)
000146  012746  000002                  MOV     #2,-(SP)
000152  010600                          MOV     SP,R0                           ; SP,*
000154  104414                          TRAP    14
000156  016700  040374'                 MOV     ML.ADDR,R0                      ;                               2212
000162  016066  000006  000044         MOV     6(R0),44(SP)                    ; *,ML.REG
000170  016616  000044                  MOV     44(SP),(SP)                     ; ML.REG,*
000174  012746  003556'                 MOV     #DA.PRINT,-(SP)
000200  012746  000002                  MOV     #2,-(SP)
000204  010600                          MOV     SP,R0                           ; SP,*
000206  104414                          TRAP    14
000210  016700  040374'                 MOV     ML.ADDR,R0                      ;                               2213
000214  016066  000010  000046         MOV     10(R0),46(SP)                   ; *,ML.REG
000222  016616  000046                  MOV     46(SP),(SP)                     ; ML.REG,*
000226  012746  003642'                 MOV     #CS2.PRINT,-(SP)
000232  012746  000002                  MOV     #2,-(SP)
000236  010600                          MOV     SP,R0                           ; SP,*
000240  104414                          TRAP    14
000242  016700  040374'                 MOV     ML.ADDR,R0                      ;                               2214
000246  016066  000012  000050         MOV     12(R0),50(SP)                   ; *,ML.REG
000254  016616  000050                  MOV     50(SP),(SP)                     ; ML.REG,*
000260  012746  003730'                 MOV     #DS.PRINT,-(SP)
000264  012746  000002                  MOV     #2,-(SP)
```

```
000270  010600                          MOV     SP,R0                           ; SP,*
000272  104414                          TRAP    14
000274  016700  040374'                 MOV     ML.ADDR,R0                      ;                       2215
000300  016066  000014  000052          MOV     14(R0),52(SP)                   ; *,ML.REG
000306  016616  000052                  MOV     52(SP),(SP)                     ; ML.REG,*
000312  012746  004014'                 MOV     #ER.PRINT,-(SP)
000316  012746  000002                  MOV     #2,-(SP)
000322  010600                          MOV     SP,R0                           ; SP,*
000324  104414                          TRAP    14
000326  016700  040374'                 MOV     ML.ADDR,R0                      ;                       2216
000332  016066  000016  000054          MOV     16(R0),54(SP)                   ; *,ML.REG
000340  016616  000054                  MOV     54(SP),(SP)                     ; ML.REG,*
000344  012746  004100'                 MOV     #AS.PRINT,-(SP)
000350  012746  000002                  MOV     #2,-(SP)
000354  010600                          MOV     SP,R0                           ; SP,*
000356  104414                          TRAP    14
000360  016700  040374'                 MOV     ML.ADDR,R0                      ;                       2217
000364  016066  000024  000056          MOV     24(R0),56(SP)                   ; *,ML.REG
000372  016616  000056                  MOV     56(SP),(SP)                     ; ML.REG,*
000376  012746  004164'                 MOV     #MR.PRINT,-(SP)
000402  012746  000002                  MOV     #2,-(SP)
000406  010600                          MOV     SP,R0                           ; SP,*
000410  104414                          TRAP    14
000412  016700  040374'                 MOV     ML.ADDR,R0                      ;                       2218
000416  016066  000026  000060          MOV     26(R0),60(SP)                   ; *,ML.REG
000424  016616  000060                  MOV     60(SP),(SP)                     ; ML.REG,*
000430  012746  004250'                 MOV     #DT.PRINT,-(SP)
000434  012746  000002                  MOV     #2,-(SP)
000440  010600                          MOV     SP,R0                           ; SP,*
000442  104414                          TRAP    14
000444  016700  040374'                 MOV     ML.ADDR,R0                      ;                       2219
000450  016066  000030  000062          MOV     30(R0),62(SP)                   ; *,ML.REG
000456  016616  000062                  MOV     62(SP),(SP)                     ; ML.REG,*
000462  012746  004334'                 MOV     #SN.PRINT,-(SP)
000466  012746  000002                  MOV     #2,-(SP)
000472  010600                          MOV     SP,R0                           ; SP,*
000474  104414                          TRAP    14
000476  016700  040374'                 MOV     ML.ADDR,R0                      ;                       2220
000502  016066  000042  000064          MOV     42(R0),64(SP)                   ; *,ML.REG
000510  016616  000064                  MOV     64(SP),(SP)                     ; ML.REG,*
000514  012746  004420'                 MOV     #EE.PRINT,-(SP)
000520  012746  000002                  MOV     #2,-(SP)
000524  010600                          MOV     SP,R0                           ; SP,*
000526  104414                          TRAP    14
000530  016700  040374'                 MOV     ML.ADDR,R0                      ;                       2221
000534  016066  000044  000066          MOV     44(R0),66(SP)                   ; *,ML.REG
000542  016616  000066                  MOV     66(SP),(SP)                     ; ML.REG,*
000546  012746  004504'                 MOV     #EL.PRINT,-(SP)
000552  012746  000002                  MOV     #2,-(SP)
000556  010600                          MOV     SP,R0                           ; SP,*
000560  104414                          TRAP    14
000562  062706  000124                  ADD     #124,SP                         ;                       2206
```

```
000566  000207                                  RTS     PC

; Routine Size:  184 words
; Maximum stack depth per invocation:  42 words
```

```
;          2223  %sbttl 'ROUTINE DECLARATIONS'
;          2224
;          2225  global routine SUMMARY : novalue =
;          2226      begin
;          2227
;          2228  !++
;          2229  ! Functional Description:
;          2230  !       This global routine is called from the report
;          2231  !       code section contained in skell 2 of this diagnostic.
;          2232  !
;          2233  !       The purpose of this routine is to report to the
;          2234  !       operator a statistical report of the prom maintenance
;          2235  !       program on this unit.
;          2236  !
;          2237  ! Formal Parameters:
;          2238  !       none
;          2239  !
;          2240  ! Implicit Inputs:
;          2241  !       WBUFF
;          2242  !
;          2243  ! Implicit Outputs:
;          2244  !
;          2245  ! Completion codes:
;          2246  !       none
;          2247  !
;          2248  ! Side Effects:
;          2249  !       none
;          2250  !--
;          2251
;          2252      !+
;          2253      ! See if any additional errors were found bad
;          2254      ! during execution of the Prom Maintenance program.
;          2255      ! If additional errors were found then dump to the
;          2256      ! the console terminal the Prom Maintenance program
;          2257      ! summary report code else print that no additional
;          2258      ! were found.
;          2259      !-
;          2260
;          2261      if .PM_COUNT gtr ZERO                         !Were additional errors found
;          2262      then
;          2263          begin
;          2264          PRINTB (CRLF);                            !Print a formating <crlf>
;          2265          PRINTB (ONE_MSG, PM_HEADER);              !Print the header message
;          2266          PRINTB (PM_10_MSG);                       !Print out summary message
;          2267          PRINTB (PM_11_MSG);                       !Print out summary message
;          2268          PRINTB (PM_12_MSG);                       !Print out summary message
;          2269          PRINTB (PM_13_MSG);                       !Print out summary message
;          2270          PRINTB (CRLF);                            !Print another formating <crlf>
;          2271          PRINTB (ONE_MSG, PM$HEADER);              !Print out the chip location header
;          2272          !
;          2273          ! Index through the summary table and print out its
;          2274          ! contents to the console terminal for operator review.
```

```
;          2275          !
;          2276
;          2277          incr index from 0 to .PM_COUNT - 1 do
;          2278              begin
;        P 2279              PRINTB (FMT16,                          !This is the printing format
;        P 2280                  .PM_LOG [.index, UNIT$_PM],         !Print the failing unit of this sbe
;        P 2281                  .PM_LOG [.index, BRD$_PM],          !Print the failing board number
;        P 2282                  .PM_LOG [.index, BNK$_PM],          !Print the failing bank number
;        P 2283                  .PM_LOG [.index, BIT$_PM],          !Print the failing bit number
;          2284                  .PM_LOG [.index, SUM$_PM]);         !Print the total rows/col blasted
;          2285              end;
;          2286
;          2287          end
;          2288      else
;          2289          begin
;          2290          PRINTB (CRLF);                              !Print a formating <crlf>
;          2291          PRINTB (ONE_MSG, PM_HEADER);                !Print the header message
;          2292          PRINTB (PM_10_MSG);                         !Print out summary message
;          2293          PRINTB (PM_11_MSG);                         !Print out summary message
;          2294          PRINTB (PM_12_MSG);                         !Print out summary message
;          2295          PRINTB (PM_13_MSG);                         !Print out summary message
;          2296          PRINTB (CRLF);                              !Print another formating <crlf>
;          2297          PRINTB (ONE_MSG, PM_14_MSG);
;          2298          end;
;          2299
;          2300      end;


                                        .SBTTL  SUMMARY ROUTINE DECLARATIONS
000570  004167  000000G        SUMMARY::
                                JSR     R1,$SAVE3                       ;                       2225
000574  005767  000000'                 TST     PM.COUNT                ;                       2261
000600  003546                          BLE     3$
000602  012746  002622'                 MOV     #CRLF,-(SP)             ;                       2264
000606  012746  000001                  MOV     #1,-(SP)
000612  010600                          MOV     SP,R0                   ; SP,*
000614  104414                          TRAP    14
000616  012716  000170'                 MOV     #PM.HEADER,(SP)         ;                       2265
000622  012746  002556'                 MOV     #ONE.MSG,-(SP)
000626  012746  000002                  MOV     #2,-(SP)
000632  010600                          MOV     SP,R0                   ; SP,*
000634  104414                          TRAP    14
000636  012716  000324'                 MOV     #PM.10.MSG,(SP)         ;                       2266
000642  012746  000001                  MOV     #1,-(SP)
000646  010600                          MOV     SP,R0                   ; SP,*
000650  104414                          TRAP    14
000652  012716  000434'                 MOV     #PM.11.MSG,(SP)         ;                       2267
000656  012746  000001                  MOV     #1,-(SP)
000662  010600                          MOV     SP,R0                   ; SP,*
000664  104414                          TRAP    14
000666  012716  000544'                 MOV     #PM.12.MSG,(SP)         ;                       2268
000672  012746  000001                  MOV     #1,-(SP)
```

```
000676  010600                          MOV     SP,R0                           ; SP,*
000700  104414                          TRAP    14
000702  012716  000656'                  MOV     #PM.13.MSG,(SP)                 ;                               2269
000706  012746  000001                  MOV     #1,-(SP)
000712  010600                          MOV     SP,R0                           ; SP,*
000714  104414                          TRAP    14
000716  012716  002622'                  MOV     #CRLF,(SP)                      ;                               2270
000722  012746  000001                  MOV     #1,-(SP)
000726  010600                          MOV     SP,R0                           ; SP,*
000730  104414                          TRAP    14
000732  012716  000104'                  MOV     #PM$HEADER,(SP)                 ;                               2271
000736  012746  002556'                  MOV     #ONE.MSG,-(SP)
000742  012746  000002                  MOV     #2,-(SP)
000746  010600                          MOV     SP,R0                           ; SP,*
000750  104414                          TRAP    14
000752  016703  000000'                  MOV     PM.COUNT,R3                     ;                               2277
000756  005002                          CLR     R2                              ; INDEX
000760  000453                          BR      2$
000762  010201                  1$:     MOV     R2,R1                           ; INDEX,*                       2284
000764  006301                          ASL     R1
000766  006301                          ASL     R1
000770  016146  000004'                  MOV     PM.LOG+2(R1),-(SP)
000774  062701  000002'                  ADD     #PM.LOG,R1
001000  011100                          MOV     (R1),R0
001002  006200                          ASR     R0
001004  006200                          ASR     R0
001006  006200                          ASR     R0
001010  006200                          ASR     R0
001012  006200                          ASR     R0
001014  006200                          ASR     R0
001016  042700  177600                  BIC     #177600,R0
001022  010046                          MOV     R0,-(SP)
001024  111146                          MOVB    (R1),-(SP)
001026  042716  177774                  BIC     #177774,(SP)
001032  111100                          MOVB    (R1),R0
001034  006200                          ASR     R0
001036  006200                          ASR     R0
001040  042700  177760                  BIC     #177760,R0
001044  010046                          MOV     R0,-(SP)
001046  011100                          MOV     (R1),R0
001050  006100                          ROL     R0
001052  006100                          ROL     R0
001054  006100                          ROL     R0
001056  006100                          ROL     R0
001060  042700  177770                  BIC     #177770,R0
001064  010046                          MOV     R0,-(SP)
001066  012746  000000'                  MOV     #FMT16,-(SP)
001072  012746  000006                  MOV     #6,-(SP)
001076  010600                          MOV     SP,R0                           ; SP,*
001100  104414                          TRAP    14
001102  062706  000016                  ADD     #16,SP                          ;                               2278
001106  005202                          INC     R2                              ; INDEX                         2277
```

```
001110  020203                      2$:   CMP   R2,R3                    ; INDEX,*
001112  002723                            BLT   1$
001114  000464                            BR    4$                      ;                    2261
001116  012746  002622'              3$:   MOV   #CRLF,-(SP)             ;                    2290
001122  012746  000001                    MOV   #1,-(SP)
001126  010600                            MOV   SP,R0                   ; SP,*
001130  104414                            TRAP  14
001132  012716  000170'                    MOV   #PM.HEADER,(SP)         ;                    2291
001136  012746  002556'                    MOV   #ONE.MSG,-(SP)
001142  012746  000002                    MOV   #2,-(SP)
001146  010600                            MOV   SP,R0                   ; SP,*
001150  104414                            TRAP  14
001152  012716  000324'                    MOV   #PM.10.MSG,(SP)         ;                    2292
001156  012746  000001                    MOV   #1,-(SP)
001162  010600                            MOV   SP,R0                   ; SP,*
001164  104414                            TRAP  14
001166  012716  000434'                    MOV   #PM.11.MSG,(SP)         ;                    2293
001172  012746  000001                    MOV   #1,-(SP)
001176  010600                            MOV   SP,R0                   ; SP,*
001200  104414                            TRAP  14
001202  012716  000544'                    MOV   #PM.12.MSG,(SP)         ;                    2294
001206  012746  000001                    MOV   #1,-(SP)
001212  010600                            MOV   SP,R0                   ; SP,*
001214  104414                            TRAP  14
001216  012716  000656'                    MOV   #PM.13.MSG,(SP)         ;                    2295
001222  012746  000001                    MOV   #1,-(SP)
001226  010600                            MOV   SP,R0                   ; SP,*
001230  104414                            TRAP  14
001232  012716  002622'                    MOV   #CRLF,(SP)              ;                    2296
001236  012746  000001                    MOV   #1,-(SP)
001242  010600                            MOV   SP,R0                   ; SP,*
001244  104414                            TRAP  14
001246  012716  000244'                    MOV   #PM.14.MSG,(SP)         ;                    2297
001252  012746  002556'                    MOV   #ONE.MSG,-(SP)
001256  012746  000002                    MOV   #2,-(SP)
001262  010600                            MOV   SP,R0                   ; SP,*
001264  104414                            TRAP  14
001266  062706  000026              4$:   ADD   #26,SP                  ;                    2226
001272  000207                            RTS   PC                      ;                    2225

; Routine Size:  162 words
; Maximum stack depth per invocation:  22 words


;        2301
```

```
;          2302  routine GEN_RANDOM_DATA : novalue =
;          2303      begin
;          2304
;          2305  !++
;          2306  ! Functional Description:
;          2307  !       This global routine will generate a write buffer
;          2308  !       of random data.  The global routine to actually
;          2309  !       generate the random number is a global routine
;          2310  !       linked at link time.
;          2311  !
;          2312  ! Formal Parameters:
;          2313  !       none
;          2314  !
;          2315  ! Implicit Inputs:
;          2316  !       WBUFF
;          2317  !
;          2318  ! Implicit Outputs:
;          2319  !       WBUFF is full up with random data
;          2320  !
;          2321  ! Completion codes:
;          2322  !       none
;          2323  !
;          2324  ! Side Effects:
;          2325  !       none
;          2326  !--
;          2327
;          2328      local
;          2329          OFFSET;                               !Offset variable into write buffer
;          2330
;          2331      WRT_BUF + %o'0' = .SEED1;                 !Initial seed1 to write buffer + 0
;          2332      WRT_BUF + %o'2' = .SEED2;                 !Initial seed2 to write buffer + 2
;          2333      WRT_BUF + %o'4' = .SEED3;                 !Initial seed3 to write buffer + 4
;          2334      OFFSET = %o'6';                           !Adjust offset by 6
;          2335  !
;          2336  ! VER CZMLCB CHANGED 128 TO 256
;          2337  !
;          2338      !
;          2339      ! Fill the write buffer up with random data
;          2340      !
;          2341
;          2342      incr WRD_CNT from 4 to (256 + 511*2) do   !Load 768 words with random data
;          2343          begin
;          2344          RANGEN ();                            !Generate the random number
;          2345          BREAK;
;          2346          WRT_BUF + .OFFSET = .RANDAT;          !Write buffer + offset gets the random number
;          2347          OFFSET = .OFFSET + 2;                 !Bump the offset by 2
;          2348          end;
;          2349
;          2350      end;

                                        .SBTTL  GEN.RANDOM.DATA ROUTINE DECLARATIONS
```

```
001274  004167  000000G                 GEN.RANDOM.DATA:
                                                JSR     R1,$SAVE2                       ;                               2302
001300  016767  000000G 002234'                 MOV     SEED1,WRT.BUF                   ;                               2331
001306  016767  000000G 002236'                 MOV     SEED2,WRT.BUF+2                 ;                               2332
001314  016767  000000G 002240'                 MOV     SEED3,WRT.BUF+4                 ;                               2333
001322  012701  000006                          MOV     #6,R1                           ; *,OFFSET                      2334
001326  012702  000004                          MOV     #4,R2                           ; *,WRD.CNT                     2342
001332  004767  000000G                 1$:     JSR     PC,RANGEN                       ;                               2344
001336  104422                                  TRAP    22
001340  016761  000000G 002234'                 MOV     RANDAT,WRT.BUF(R1)              ; *,*(OFFSET)                   2346
001346  062701  000002                          ADD     #2,R1                           ; *,OFFSET                      2347
001352  005202                                  INC     R2                              ; WRD.CNT                       2342
001354  020227  002376                          CMP     R2,#2376                        ; WRD.CNT,*
001360  003764                                  BLE     1$
001362  000207                                  RTS     PC                              ;                               2302

; Routine Size:  28 words
; Maximum stack depth per invocation:  3 words
```

```
;          2351  routine DM_RD_TRANSFER (SIZE, DST, SRC) : novalue =
;          2352      begin
;          2353
;          2354  !++
;          2355  ! Functional Description:
;          2356  !       This global routine will perfrom a diagnostic
;          2357  !       mode read transfer at the designated
;          2358  !       buffer and sector addresses.
;          2359  !
;          2360  ! Formal Parameters:
;          2361  !       SIZE:
;          2362  !               Specifies the size of the transfer
;          2363  !       BUFFER:
;          2364  !               Write buffer address where data is
;          2365  !               coming from or going to.
;          2366  !       SECTOR:
;          2367  !               MiL-11 desired sector address where data
;          2368  !               is coming from or going to.
;          2369  !
;          2370  ! Implicit Inputs:
;          2371  !       none
;          2372  !
;          2373  ! Implicit Outputs:
;          2374  !       none
;          2375  !
;          2376  ! Completion codes:
;          2377  !       none
;          2378  !
;          2379  ! Side Effects:
;          2380  !       none
;          2381  !--
;          2382
;          2383      CLR_MBUS;                                   !Clear the mass bus
;          2384      DAT_DM (ENABE);                             !Enable data diag mode
;          2385      WRT_RH (MLCS2, DRV_SEL, .ML_DUT);           !Select the divice under test
;          2386      WRT_RH (MLWC, WC_REG, .SIZE);               !Load the transfer size
;          2387      WRT_RH (MLDA, DA_REG, .SRC);                !Load the transfer source address
;          2388      WRT_RH (MLBA, BA_REG, .DST);                !Load the transfer destination address
;          2389      WRT_RH (MLCS1, FUNC, FUNC_4);               !Load a read function with go set
;          2390      end;


                                        .SBTTL  DM.RD.TRANSFER ROUTINE DECLARATIONS
001364  010146                  DM.RD.TRANSFER:
                                        MOV     R1,-(SP)                ;                       2351
001366  016700  040374'                 MOV     ML.ADDR,R0              ;                       2352
001372  016046  000010                  MOV     10(R0),-(SP)            ; *,ML.REG
001376  011601                          MOV     (SP),R1                 ; ML.REG,MLREG
001400  152701  000040                  BISB    #40,R1                  ; *,MLREG
001404  016700  040374'                 MOV     ML.ADDR,R0
001410  010160  000010                  MOV     R1,10(R0)               ; MLREG,*
001414  016700  040374'                 MOV     ML.ADDR,R0
```

```
001420  016046  000010          MOV     10(R0),-(SP)                    ; *,ML.REG
001424  011601                  MOV     (SP),R1                         ; ML.REG,MLREG
001426  016700  040400'        MOV     ML.DUT,R0
001432  042700  177770          BIC     #177770,R0
001436  142701  000007          BICB    #7,R1                           ; *,MLREG
001442  050001                  BIS     R0,R1                           ; *,MLREG
001444  016700  040374'        MOV     ML.ADDR,R0
001450  010160  000010          MOV     R1,10(R0)                       ; MLREG,*
001454  016700  040374'        MOV     ML.ADDR,R0                      ;                       2384
001460  016046  000024          MOV     24(R0),-(SP)                    ; *,ML.REG
001464  011601                  MOV     (SP),R1                         ; ML.REG,MLREG
001466  152701  000010          BISB    #10,R1                          ; *,MLREG
001472  016700  040374'        MOV     ML.ADDR,R0
001476  010160  000024          MOV     R1,24(R0)                       ; MLREG,*
001502  016700  040374'        MOV     ML.ADDR,R0                      ;                       2385
001506  016046  000010          MOV     10(R0),-(SP)                    ; *,ML.REG
001512  011601                  MOV     (SP),R1                         ; ML.REG,MLREG
001514  016700  040400'        MOV     ML.DUT,R0
001520  042700  177770          BIC     #177770,R0
001524  142701  000007          BICB    #7,R1                           ; *,MLREG
001530  050001                  BIS     R0,R1                           ; *,MLREG
001532  016700  040374'        MOV     ML.ADDR,R0
001536  010160  000010          MOV     R1,10(R0)                       ; MLREG,*
001542  016700  040374'        MOV     ML.ADDR,R0                      ;                       2386
001546  016046  000002          MOV     2(R0),-(SP)                     ; *,ML.REG
001552  016601  000022          MOV     22(SP),R1                       ; SIZE,MLREG
001556  016700  040374'        MOV     ML.ADDR,R0
001562  010160  000002          MOV     R1,2(R0)                        ; MLREG,*
001566  016700  040374'        MOV     ML.ADDR,R0                      ;                       2387
001572  016046  000006          MOV     6(R0),-(SP)                     ; *,ML.REG
001576  016601  000020          MOV     20(SP),R1                       ; SRC,MLREG
001602  016700  040374'        MOV     ML.ADDR,R0
001606  010160  000006          MOV     R1,6(R0)                        ; MLREG,*
001612  016700  040374'        MOV     ML.ADDR,R0                      ;                       2388
001616  016046  000004          MOV     4(R0),-(SP)                     ; *,ML.REG
001622  016601  000024          MOV     24(SP),R1                       ; DST,MLREG
001626  016700  040374'        MOV     ML.ADDR,R0
001632  010160  000004          MOV     R1,4(R0)                        ; MLREG,*
001636  017746  040374'        MOV     @ML.ADDR,-(SP)                  ; *,ML.REG              2389
001642  012600                  MOV     (SP)+,R0                        ; ML.REG,MLREG
001644  142700  000077          BICB    #77,R0                          ; *,MLREG
001650  152700  000051          BISB    #51,R0                          ; *,MLREG
001654  010077  040374'        MOV     R0,@ML.ADDR                     ; MLREG,*
001660  062706  000016          ADD     #16,SP                          ;                       2351
001664  012601                  MOV     (SP)+,R1
001666  000207                  RTS     PC

; Routine Size:  98 words
; Maximum stack depth per invocation:  10 words
```

```
;         2391  routine DM_WRT_TRANSFER (SIZE, DST, SRC) : novalue =
;         2392      begin
;         2393
;         2394  !++
;         2395  ! Functional Description:
;         2396  !       This global routine will perform a diagnostic
;         2397  !       mode write transfer at the designated
;         2398  !       buffer and sector addresses.
;         2399  !
;         2400  ! Formal Parameters:
;         2401  !       SIZE:
;         2402  !               Size of the transfer
;         2403  !       BUFFER:
;         2404  !               Write buffer address where data is
;         2405  !               coming from or going to.
;         2406  !       SECTOR:
;         2407  !               ML-11 desired sector address where data
;         2408  !               is coming from or going to.
;         2409  !
;         2410  ! Implicit Inputs:
;         2411  !       none
;         2412  !
;         2413  ! Implicit Outputs:
;         2414  !       none
;         2415  !
;         2416  ! Completion codes:
;         2417  !       none
;         2418  !
;         2419  ! Side Effects:
;         2420  !       none
;         2421  !--
;         2422
;         2423      CLR_MBUS;                                   !Clear the mass bus
;         2424      DAT_DM (ENABE);                             !Enable data diag mode
;         2425      WRT_RH (MLCS2, DRV_SEL, .ML_DUT);           !Select the divice under test
;         2426      WRT_RH (MLWC, WC_REG, .SIZE);               !Load the transfer size
;         2427      WRT_RH (MLDA, DA_REG, .DST);                !Load the destination address
;         2428      WRT_RH (MLBA, BA_REG, .SRC);                !Load the source address
;         2429      WRT_RH (MLCS1, FUNC, FUNC_5);               !Load a write function with go set
;         2430      end;


                                .SBTTL  DM.WRT.TRANSFER ROUTINE DECLARATIONS
001670  010146          DM.WRT.TRANSFER:
                        MOV     R1,-(SP)                ;                                       2391
001672  016700  040374'  MOV     ML.ADDR,R0              ;                                       2392
001676  016046  000010    MOV     10(R0),-(SP)            ; *,ML.REG
001702  011601            MOV     (SP),R1                 ; ML.REG,MLREG
001704  152701  000040    BISB    #40,R1                  ; *,MLREG
001710  016700  040374'  MOV     ML.ADDR,R0
001714  010160  000010    MOV     R1,10(R0)               ; MLREG,*
001720  016700  040374'  MOV     ML.ADDR,R0
```

```
001724  016046  000010          MOV     10(R0),-(SP)                    ; *,ML.REG
001730  011601                  MOV     (SP),R1                         ; ML.REG,MLREG
001732  016700  040400'         MOV     ML.DUT,R0
001736  042700  177770          BIC     #177770,R0
001742  142701  000007          BICB    #7,R1                           ; *,MLREG
001746  050001                  BIS     R0,R1                           ; *,MLREG
001750  016700  040374'         MOV     ML.ADDR,R0
001754  010160  000010          MOV     R1,10(R0)                       ; MLREG,*
001760  016700  040374'         MOV     ML.ADDR,R0                      ;                                       2424
001764  016046  000024          MOV     24(R0),-(SP)                    ; *,ML.REG
001770  011601                  MOV     (SP),R1                         ; ML.REG,MLREG
001772  152701  000010          BISB    #10,R1                          ; *,MLREG
001776  016700  040374'         MOV     ML.ADDR,R0
002002  010160  000024          MOV     R1,24(R0)                       ; MLREG,*
002006  016700  040374'         MOV     ML.ADDR,R0                      ;                                       2425
002012  016046  000010          MOV     10(R0),-(SP)                    ; *,ML.REG
002016  011601                  MOV     (SP),R1                         ; ML.REG,MLREG
002020  016700  040400'         MOV     ML.DUT,R0
002024  042700  177770          BIC     #177770,R0
002030  142701  000007          BICB    #7,R1                           ; *,MLREG
002034  050001                  BIS     R0,R1                           ; *,MLREG
002036  016700  040374'         MOV     ML.ADDR,R0
002042  010160  000010          MOV     R1,10(R0)                       ; MLREG,*
002046  016700  040374'         MOV     ML.ADDR,R0                      ;                                       2426
002052  016046  000002          MOV     2(R0),-(SP)                     ; *,ML.REG
002056  016601  000022          MOV     22(SP),R1                       ; SIZE,MLREG
002062  016700  040374'         MOV     ML.ADDR,R0
002066  010160  000002          MOV     R1,2(R0)                        ; MLREG,*
002072  016700  040374'         MOV     ML.ADDR,R0                      ;                                       2427
002076  016046  000006          MOV     6(R0),-(SP)                     ; *,ML.REG
002102  016601  000022          MOV     22(SP),R1                       ; DST,MLREG
002106  016700  040374'         MOV     ML.ADDR,R0
002112  010160  000006          MOV     R1,6(R0)                        ; MLREG,*
002116  016700  040374'         MOV     ML.ADDR,R0                      ;                                       2428
002122  016046  000004          MOV     4(R0),-(SP)                     ; *,ML.REG
002126  016601  000022          MOV     22(SP),R1                       ; SRC,MLREG
002132  016700  040374'         MOV     ML.ADDR,R0
002136  010160  000004          MOV     R1,4(R0)                        ; MLREG,*
002142  017746  040374'         MOV     @ML.ADDR,-(SP)                  ; *,ML.REG                              2429
002146  012600                  MOV     (SP)+,R0                        ; ML.REG,MLREG
002150  142700  000077          BICB    #77,R0                          ; *,MLREG
002154  152700  000061          BISB    #61,R0                          ; *,MLREG
002160  010077  040374'         MOV     R0,@ML.ADDR                     ; MLREG,*
002164  062706  000016          ADD     #16,SP                          ;                                       2391
002170  012601                  MOV     (SP)+,R1
002172  000207                  RTS     PC

; Routine Size:  98 words
; Maximum stack depth per invocation:  10 words
```

```
;          2431  routine DM_RAND_LOAD (TEMP_ARR$BNK_SEL) : novalue =
;          2432      begin
;          2433
;          2434  !++
;          2435  ! Functional Description:
;          2436  !       This global routine will load via diagnostic mode
;          2437  !       random data to the designated array and bank
;          2438  !       selected by arr$bnk_sel.
;          2439  !
;          2440  ! Formal Parameters:
;          2441  !       ARR$BNK_SEL:
;          2442  !               This argument stores the array and
;          2443  !               bank select address.
;          2444  !
;          2445  ! Implicit Inputs:
;          2446  !       WRT_BUF, RET_STATUS
;          2447  !
;          2448  ! Implicit Outputs:
;          2449  !       WRT_BUF is filled with random data
;          2450  !
;          2451  ! Completion codes:
;          2452  !       none
;          2453  !
;          2454  ! Side Effects:
;          2455  !       none
;          2456  !--
;          2457
;          2458      local
;          2459          S_BUF_INDEX,                        !Start buffer index pointer
;          2460          BUF_INDEX;                          !Buffer index pointer
;          2461
;          2462      GEN_RANDOM_DATA ();                     !Generate a write buffer with random data
;          2463      S_BUF_INDEX = ZERO;                     !The first sector starts at write buf word zero
;          2464      BUF_INDEX = .S_BUF_INDEX;               !Buf index will get 128 random data words
;          2465      DM_WRT_TRANSFER (SIZE = -256, DST = .TEMP_ARR$BNK_SEL, SRC = WRT_BUF);
;          2466
;          2467      !+
;          2468      ! Increment through all sectors and load them
;          2469      ! with random data.
;          2470      !-
;          2471
;          2472      incr SEC_CNT from 0 to .BNK_NUM_SEC do      !Load all sectors in bank with random data
;          2473          begin
;          2474          BREAK;
;          2475
;          2476          incr WRD_CNT from 0 to 127 do           !Load all words in sector with random data
;          2477              begin
;          2478              WRT_RH (MLD1, DB1_REG, .WRT_BUF [.BUF_INDEX, WRD]); !Load data reg 1
;          2479              WRT_RH (MLD2, DB2_REG, .WRT_BUF [.BUF_INDEX, WRD]); !Load data reg 2
;          2480              WRT_RH (MLE2, E2_REG, .WRT_BUF [.BUF_INDEX, WRD]);  !Load data reg 3
;          2481              DAT_CLK;                        !Clock the random data into the ml-11 long word
;          2482              BUF_INDEX = .BUF_INDEX + ONE;   !Get the next random data word
```

```
;          2483                end;
;          2484
;          2485            S_BUF_INDEX = .S_BUF_INDEX + 2;          !Start the next sector 2 word deeper
;          2486            BUF_INDEX = .S_BUF_INDEX;                !Buf index will get 128 more random data words
;          2487            end;
;          2488
;          2489        CLR_MBUS;                                    !Clear the single step dma mode
;          2490        end;


                                        .SBTTL  DM.RAND.LOAD ROUTINE DECLARATIONS
002174  004167  000000G          DM.RAND.LOAD:
                                        JSR     R1,$SAVE5                       ;                               2431
002200  162706  000020                  SUB     #20,SP
002204  004767  177064                  JSR     PC,GEN.RANDOM.DATA              ;                               2462
002210  005005                          CLR     R5                              ; S.BUF.INDEX                   2463
002212  005004                          CLR     R4                              ; BUF.INDEX                     2464
002214  012746  177400                  MOV     #-400,-(SP)                     ;                               2465
002220  011667  040426'                 MOV     (SP),SIZE
002224  016646  000040                  MOV     40(SP),-(SP)                    ; TEMP.ARR$BNK.SE,*
002230  011667  040430'                 MOV     (SP),DST
002234  012746  002234'                 MOV     #WRT.BUF,-(SP)
002240  011667  040432'                 MOV     (SP),SRC
002244  004767  177420                  JSR     PC,DM.WRT.TRANSFER
002250  016766  040416'  000010         MOV     BNK.NUM.SEC,10(SP)              ;                               2472
002256  005066  000006                  CLR     6(SP)                           ; SEC.CNT
002262  000474                          BR      3$
002264  104422                  1$:     TRAP    22                              ;                               2473
002266  005000                          CLR     R0                              ; WRD.CNT                       2476
002270  016703  040374'          2$:     MOV     ML.ADDR,R3                      ;                               2478
002274  016366  000036  000024          MOV     36(R3),24(SP)                   ; *,ML.REG
002302  010403                          MOV     R4,R3                           ; BUF.INDEX,*
002304  006303                          ASL     R3
002306  012701  002234'                 MOV     #WRT.BUF,R1
002312  060301                          ADD     R3,R1
002314  011102                          MOV     (R1),R2                         ; *,MLREG
002316  016703  040374'                 MOV     ML.ADDR,R3
002322  010263  000036                  MOV     R2,36(R3)                       ; MLREG,*
002326  016703  040374'                 MOV     ML.ADDR,R3                      ;                               2479
002332  016366  000040  000022         MOV     40(R3),22(SP)                   ; *,ML.REG
002340  011102                          MOV     (R1),R2                         ; *,MLREG
002342  016703  040374'                 MOV     ML.ADDR,R3
002346  010263  000040                  MOV     R2,40(R3)                       ; MLREG,*
002352  016703  040374'                 MOV     ML.ADDR,R3                      ;                               2480
002356  016366  000034  000020         MOV     34(R3),20(SP)                   ; *,ML.REG
002364  011102                          MOV     (R1),R2                         ; *,MLREG
002366  016703  040374'                 MOV     ML.ADDR,R3
002372  010263  000034                  MOV     R2,34(R3)                       ; MLREG,*
002376  016703  040374'                 MOV     ML.ADDR,R3
002402  016366  000024  000016         MOV     24(R3),16(SP)                   ; *,ML.REG
002410  016602  000016                  MOV     16(SP),R2                       ; ML.REG,MLREG
002414  152702  000020                  BISB    #20,R2                          ; *,MLREG
```

```
002420  016703  040374'                        MOV     ML.ADDR,R3
002424  010263  000024                        MOV     R2,24(R3)                       ; MLREG,*
002430  005204                                INC     R4                              ; BUF.INDEX                     2482
002432  005200                                INC     R0                              ; WRD.CNT                       2476
002434  020027  000177                        CMP     R0,#177                         ; WRD.CNT,*
002440  003713                                BLE     2$
002442  062705  000002                        ADD     #2,R5                           ; *,S.BUF.INDEX                 2485
002446  010504                                MOV     R5,R4                           ; S.BUF.INDEX,BUF.INDEX         2486
002450  005266  000006                        INC     6(SP)                           ; SEC.CNT                       2472
002454  026666  000006  000010        3$:     CMP     6(SP),10(SP)                    ; SEC.CNT,*
002462  003700                                BLE     1$
002464  016703  040374'                        MOV     ML.ADDR,R3                      ;                               2487
002470  016366  000010  000014                MOV     10(R3),14(SP)                   ; *,ML.REG
002476  016602  000014                        MOV     14(SP),R2                       ; ML.REG,MLREG
002502  152702  000040                        BISB    #40,R2                          ; *,MLREG
002506  016703  040374'                        MOV     ML.ADDR,R3
002512  010263  000010                        MOV     R2,10(R3)                       ; MLREG,*
002516  016703  040374'                        MOV     ML.ADDR,R3
002522  016366  000010  000012                MOV     10(R3),12(SP)                   ; *,ML.REG
002530  016602  000012                        MOV     12(SP),R2                       ; ML.REG,MLREG
002534  016705  040400'                        MOV     ML.DUT,R5
002540  042705  177770                        BIC     #177770,R5
002544  142702  000007                        BICB    #7,R2                           ; *,MLREG
002550  050502                                BIS     R5,R2                           ; *,MLREG
002552  016703  040374'                        MOV     ML.ADDR,R3
002556  010263  000010                        MOV     R2,10(R3)                       ; MLREG,*
002562  062706  000026                        ADD     #26,SP                          ;                               2431
002566  000207                                RTS     PC

; Routine Size:  126 words
; Maximum stack depth per invocation:  17 words
```

```
;          2491  routine DM_1_0_LOAD (TEMP_ARR$BNK_SEL, DATA) : novalue =
;          2492      begin
;          2493
;          2494  !++
;          2495  ! Functional Description:
;          2496  !       This global routine will load via diagnostic mode
;          2497  !       ones or ZEROS " the contents of data " to
;          2498  !       the designated array and bank selected by arr$bnk_sel.
;          2499  !
;          2500  ! Formal Parameters:
;          2501  !       ARR$BNK_SEL:
;          2502  !               This argument stores the array and bank
;          2503  !               select address.
;          2504  !       DATA:
;          2505  !               This argument contains the selected data pattern
;          2506  !               to be written into the array and bank selected by
;          2507  !               array bank select.
;          2508  !
;          2509  ! Implicit Inputs:
;          2510  !       WRT_BUF, RET_STATUS
;          2511  !
;          2512  ! Implicit Outputs:
;          2513  !       WRT_BUF is filled with ones or zeros
;          2514  !       data.
;          2515  !
;          2516  ! Completion codes:
;          2517  !       none
;          2518  !
;          2519  ! Side Effects:
;          2520  !       none
;          2521  !--
;          2522
;          2523                                                !Single step dma write mode
;          2524      DM_WRT_TRANSFER (SIZE = -256, DST = .TEMP_ARR$BNK_SEL, SRC = WRT_BUF);
;          2525      !
;          2526      ! Load the diagnostic registers up with the
;          2527      ! data contained in the argument 'data'. Data
;          2528      ! will either be all ones or all zeros.
;          2529      !
;          2530      WRT_RH (MLD1, DB1_REG, .DATA);                !Load data reg 1 with data
;          2531      WRT_RH (MLD2, DB2_REG, .DATA);                !Load data reg 2 with data
;          2532      WRT_RH (MLE2, E2_REG, .DATA);                 !Load data reg 3 with data
;          2533
;          2534      !+
;          2535      ! Load this banks sector range up with
;          2536      ! the data just loaded into the diagnostic
;          2537      ! registers.  The outer loop will select
;          2538      ! all the sectors within this bank while
;          2539      ! the inner loop will load the data into
;          2540      ! all the words within the sector.
;          2541      !-
;          2542
```

```
;          2543      incr SEC_CNT from 0 to .BNK_NUM_SEC do      !Load all sectors in bank with data
;          2544
;          2545          incr WRD_CNT from 0 to 127 do           !Load all words in sector with data
;          2546              DAT_CLK;                            !Clock the data into the ml-11
;          2547
;          2548      !
;          2549      ! VER CZMLCB changed 127 to 255
;          2550      !
;          2551
;          2552          !+
;          2553          ! Fill the write buffer up with the same
;          2554          ! data pattern that was just transfered
;          2555          ! to the unit under test.
;          2556          !
;          2557          ! This will later be used as compare data
;          2558          ! when this sector number is read back.
;          2559          !-
;          2560
;          2561          incr WRD_CNT from 0 to (255 + 511*2) do     !Fill the write buffer up with data
;          2562              WRT_BUF [.WRD_CNT, WRD] = .DATA;
;          2563
;          2564          CLR_MBUS;                                   !Clear the single step dma mode
;          2565          end;


                                        .SBTTL  DM.1.0.LOAD ROUTINE DECLARATIONS
002570  004167  000000G          DM.1.0.LOAD:
                                        JSR     R1,$SAVE5                       ;                                   2491
002574  162706  000014                  SUB     #14,SP
002600  012746  177400                  MOV     #-400,-(SP)                     ;                                   2524
002604  011667  040426'                 MOV     (SP),SIZE
002610  016646  000036                  MOV     36(SP),-(SP)                    ; TEMP.ARR$BNK.SE,*
002614  011667  040430'                 MOV     (SP),DST
002620  012746  002234'                 MOV     #WRT.BUF,-(SP)
002624  011667  040432'                 MOV     (SP),SRC
002630  004767  177034                  JSR     PC,DM.WRT.TRANSFER
002634  016701  040374'                 MOV     ML.ADDR,R1                      ;                                   2530
002640  016166  000036  000020         MOV     36(R1),20(SP)                   ; *,ML.REG
002646  016605  000040                  MOV     40(SP),R5                       ; DATA,*
002652  010500                          MOV     R5,R0                           ; *,MLREG
002654  016701  040374'                 MOV     ML.ADDR,R1
002660  010061  000036                  MOV     R0,36(R1)                       ; MLREG,*
002664  016701  040374'                 MOV     ML.ADDR,R1                      ;                                   2531
002670  016166  000040  000016         MOV     40(R1),16(SP)                   ; *,ML.REG
002676  016701  040374'                 MOV     ML.ADDR,R1
002702  010061  000040                  MOV     R0,40(R1)                       ; MLREG,*
002706  016701  040374'                 MOV     ML.ADDR,R1                      ;                                   2532
002712  016166  000034  000014         MOV     34(R1),14(SP)                   ; *,ML.REG
002720  016701  040374'                 MOV     ML.ADDR,R1
002724  010061  000034                  MOV     R0,34(R1)                       ; MLREG,*
002730  016704  040416'                 MOV     BNK.NUM.SEC,R4                  ;                                   2543
002734  005003                          CLR     R3                              ; SEC.CNT
```

```
002736  000423                          BR      3$
002740  005002                  1$:     CLR     R2                      ; WRD.CNT                       2545
002742  016701  040374'          2$:     MOV     ML.ADDR,R1
002746  016166  000024  000012          MOV     24(R1),12(SP)           ; *,ML.REG
002754  016600  000012                  MOV     12(SP),R0               ; ML.REG,MLREG
002760  152700  000020                  BISB    #20,R0                  ; *,MLREG
002764  016701  040374'                  MOV     ML.ADDR,R1
002770  010061  000024                  MOV     R0,24(R1)               ; MLREG,*
002774  005202                          INC     R2                      ; WRD.CNT
002776  020227  000177                  CMP     R2,#177                 ; WRD.CNT,*
003002  003757                          BLE     2$
003004  005203                          INC     R3                      ; SEC.CNT                       2543
003006  020304                  3$:     CMP     R3,R4                   ; SEC.CNT,*
003010  003753                          BLE     1$
003012  005000                          CLR     R0                      ; WRD.CNT                       2561
003014  010001                  4$:     MOV     R0,R1                   ; WRD.CNT,*                     2562
003016  006301                          ASL     R1
003020  010561  002234'                  MOV     R5,WRT.BUF(R1)
003024  005200                          INC     R0                      ; WRD.CNT                       2561
003026  020027  002375                  CMP     R0,#2375                ; WRD.CNT,*
003032  003770                          BLE     4$
003034  016701  040374'                  MOV     ML.ADDR,R1              ;                               2562
003040  016166  000010  000010          MOV     10(R1),10(SP)           ; *,ML.REG
003046  016600  000010                  MOV     10(SP),R0               ; ML.REG,MLREG
003052  152700  000040                  BISB    #40,R0                  ; *,MLREG
003056  016701  040374'                  MOV     ML.ADDR,R1
003062  010061  000010                  MOV     R0,10(R1)               ; MLREG,*
003066  016701  040374'                  MOV     ML.ADDR,R1
003072  016166  000010  000006          MOV     10(R1),6(SP)            ; *,ML.REG
003100  016600  000006                  MOV     6(SP),R0                ; ML.REG,MLREG
003104  016705  040400'                  MOV     ML.DUT,R5
003110  042705  177770                  BIC     #177770,R5
003114  142700  000007                  BICB    #7,R0                   ; *,MLREG
003120  050500                          BIS     R5,R0                   ; *,MLREG
003122  016701  040374'                  MOV     ML.ADDR,R1
003126  010061  000010                  MOV     R0,10(R1)               ; MLREG,*
003132  062706  000022                  ADD     #22,SP                  ;                               2491
003136  000207                          RTS     PC

; Routine Size:  116 words
; Maximum stack depth per invocation:  15 words
```

```
;          2566  routine WRT_CHK_TRANSFER (SIZE, DST, SRC) : novalue =
;          2567      begin
;          2568
;          2569  !++
;          2570  ! Functional Description:
;          2571  !       This global routine will perform a write check transfer
;          2572  !       at the designated buffer and sector address.
;          2573  !
;          2574  ! Formal Parameters:
;          2575  !       SIZE:
;          2576  !               Stores the size of the transfer
;          2577  !       BUFFER:
;          2578  !               Write buffer address where data is
;          2579  !               coming from or going to.
;          2580  !       SECTOR:
;          2581  !               ML-11 desired sector address where data
;          2582  !               is coming from or going to.
;          2583  !
;          2584  ! Implicit Inputs:
;          2585  !       none
;          2586  !
;          2587  ! Implicit Outputs:
;          2588  !       none
;          2589  !
;          2590  ! Completion codes:
;          2591  !       none
;          2592  !
;          2593  ! Side Effects:
;          2594  !       none
;          2595  !--
;          2596
;          2597      CLR_MBUS;                                   !Clear the mass bus
;          2598      ECC_DIS (ENABE);                            !Disable error correction
;          2599      WRT_RH (MLCS2, DRV_SEL, .ML_DUT);           !Select the divice under test
;          2600      WRT_RH (MLWC, WC_REG, .SIZE);               !Load the transfer size
;          2601      WRT_RH (MLDA, DA_REG, .DST);                !Load the destination address
;          2602      WRT_RH (MLBA, BA_REG, .SRC);                !Load the source address
;          2603      WRT_RH (MLCS1, FUNC, FUNC_4);               !Load a write check function with go set
;          2604
;          2605      if .ML_ADDR [MLCS1, SC]                     !Does the transfer cause a sc error
;          2606      then
;          2607          begin
;          2608
;          2609          if not .ML_ADDR [MLCS2, WCE]            !We expect the write check bit to be set
;          2610          then
;          2611              begin
;          2612              ERRSF (ERR_13, UNX_DRV_ERR_MSG, DUMPER);    !Report the unexpected error
;          2613              PRINTB (ONE_MSG, WT_CHK_XFER_MSG);  !Print what transfer caused the error
;          2614              DODU (.L$UNIT);                     !Drop this unit
;          2615              DOCLN;                              !Jump to the clean up code
;          2616              end;
;          2617
```

```
;          2618          end;
;          2619
;          2620      do                                          !Delay for transfer to complete
;          2621          0
;          2622      until .ML_ADDR [MLDS, DRY];
;          2623
;          2624      if .ML_ADDR [MLCS1, SC]                     !Did the transfer cause a sc error
;          2625      then
;          2626          begin
;          2627
;          2628          if not .ML_ADDR [MLCS2, WCE]            !We expect the write check bit to be set
;          2629          then
;          2630              begin
;          2631              ERRSF (ERR_13, UNX_DRV_ERR_MSG, DUMPER);    !Report the unexpected error
;          2632              PRINTB (ONE_MSG, WT_CHK_XFER_MSG);  !Print which function caused the error
;          2633              DODU (.L$UNIT);                     !Drop this unit
;          2634              DOCLN;                              !Jump to the clean up code
;          2635              end;
;          2636
;          2637          end;
;          2638
;          2639      end;


                                .SBTTL  WRT.CHK.TRANSFE ROUTINE DECLARATIONS
003140  010146                  WRT.CHK.TRANSFE:
                                MOV     R1,-(SP)                    ;                                   2566
003142  162706  000032          SUB     #32,SP
003146  016700  040374'         MOV     ML.ADDR,R0                  ;                                   2567
003152  016066  000010  000030  MOV     10(R0),30(SP)               ; *,ML.REG
003160  016601  000030          MOV     30(SP),R1                   ; ML.REG,MLREG
003164  152701  000040          BISB    #40,R1                      ; *,MLREG
003170  016700  040374'         MOV     ML.ADDR,R0
003174  010160  000010          MOV     R1,10(R0)                   ; MLREG,*
003200  016700  040374'         MOV     ML.ADDR,R0
003204  016066  000010  000026  MOV     10(R0),26(SP)               ; *,ML.REG
003212  016601  000026          MOV     26(SP),R1                   ; ML.REG,MLREG
003216  016700  040400'         MOV     ML.DUT,R0
003222  042700  177770          BIC     #177770,R0
003226  142701  000007          BICB    #7,R1                       ; *,MLREG
003232  050001                  BIS     R0,R1                       ; *,MLREG
003234  016700  040374'         MOV     ML.ADDR,R0
003240  010160  000010          MOV     R1,10(R0)                   ; MLREG,*
003244  016700  040374'         MOV     ML.ADDR,R0                  ;                                   2598
003250  016066  000024  000024  MOV     24(R0),24(SP)               ; *,ML.REG
003256  016601  000024          MOV     24(SP),R1                   ; ML.REG,MLREG
003262  152701  000002          BISB    #2,R1                       ; *,MLREG
003266  016700  040374'         MOV     ML.ADDR,R0
003272  010160  000024          MOV     R1,24(R0)                   ; MLREG,*
003276  016700  040374'         MOV     ML.ADDR,R0                  ;                                   2599
003302  016066  000010  000022  MOV     10(R0),22(SP)               ; *,ML.REG
003310  016601  000022          MOV     22(SP),R1                   ; ML.REG,MLREG
```

```
003314  016700  040400'                MOV     ML.DUT,R0
003320  042700  177770                 BIC     #177770,R0
003324  142701  000007                 BICB    #7,R1                   ; *,MLREG
003330  050001                         BIS     R0,R1                   ; *,MLREG
003332  016700  040374'                MOV     ML.ADDR,R0
003336  010160  000010                 MOV     R1,10(R0)               ; MLREG,*
003342  016700  040374'                MOV     ML.ADDR,R0              ;                       2600
003346  016066  000002  000020         MOV     2(R0),20(SP)            ; *,ML.REG
003354  016601  000042                 MOV     42(SP),R1               ; SIZE,MLREG
003360  016700  040374'                MOV     ML.ADDR,R0
003364  010160  000002                 MOV     R1,2(R0)                ; MLREG,*
003370  016700  040374'                MOV     ML.ADDR,R0              ;                       2601
003374  016066  000006  000016         MOV     6(R0),16(SP)            ; *,ML.REG
003402  016601  000040                 MOV     40(SP),R1               ; DST,MLREG
003406  016700  040374'                MOV     ML.ADDR,R0
003412  010160  000006                 MOV     R1,6(R0)                ; MLREG,*
003416  016700  040374'                MOV     ML.ADDR,R0              ;                       2602
003422  016066  000004  000014         MOV     4(R0),14(SP)            ; *,ML.REG
003430  016601  000036                 MOV     36(SP),R1               ; SRC,MLREG
003434  016700  040374'                MOV     ML.ADDR,R0
003440  010160  000004                 MOV     R1,4(R0)                ; MLREG,*
003444  017766  040374'  000012        MOV     @ML.ADDR,12(SP)         ; *,ML.REG              2603
003452  016600  000012                 MOV     12(SP),R0               ; ML.REG,MLREG
003456  142700  000077                 BICB    #77,R0                  ; *,MLREG
003462  152700  000051                 BISB    #51,R0                  ; *,MLREG
003466  010077  040374'                MOV     R0,@ML.ADDR             ; MLREG,*
003472  017766  040374'  000010        MOV     @ML.ADDR,10(SP)         ; *,ML.REG              2605
003500  100033                         BPL     1$
003502  016700  040374'                MOV     ML.ADDR,R0              ;                       2609
003506  016066  000010  000006         MOV     10(R0),6(SP)            ; *,ML.REG
003514  032766  040000  000006         BIT     #40000,6(SP)            ; *,ML.REG
003522  001022                         BNE     1$
003524  104454                         TRAP    54                      ;                       2612
003526  000015                         .WORD   15
003530  002310'                        .WORD   UNX.DRV.ERR.MSG
003532  000000'                        .WORD   DUMPER
003534  012746  002474'                MOV     #WT.CHK.XFER.MSG,-(SP)  ;                       2613
003540  012746  002556'                MOV     #ONE.MSG,-(SP)
003544  012746  000002                 MOV     #2,-(SP)
003550  010600                         MOV     SP,R0                   ; SP,*
003552  104414                         TRAP    14
003554  016700  000000G                MOV     LSUNIT,R0               ;                       2614
003560  104451                         TRAP    51
003562  104444                         TRAP    44
003564  062706  000006                 ADD     #6,SP                   ;                       2611
003570  016700  040374'        1$:     MOV     ML.ADDR,R0              ;                       2622
003574  016066  000012  000004         MOV     12(R0),4(SP)            ; *,ML.REG
003602  105766  000004                 TSTB    4(SP)                   ; ML.REG
003606  100370                         BPL     1$
003610  017766  040374'  000002        MOV     @ML.ADDR,2(SP)          ; *,ML.REG              2624
003616  100031                         BPL     2$
003620  016700  040374'                MOV     ML.ADDR,R0              ;                       2628
```

```
003624  016016  000010                  MOV     10(R0),(SP)                     ; *,ML.REG
003630  032716  040000                  BIT     #40000,(SP)                     ; *,ML.REG
003634  001022                          BNE     2$
003636  104454                          TRAP    54                              ;                                       2631
003640  000015                          .WORD   15
003642  002310'                         .WORD   UNX.DRV.ERR.MSG
003644  000000'                         .WORD   DUMPER
003646  012746  002474'                 MOV     #WT.CHK.XFER.MSG,-(SP)          ;                                       2632
003652  012746  002556'                 MOV     #ONE.MSG,-(SP)
003656  012746  000002                  MOV     #2,-(SP)
003662  010600                          MOV     SP,R0                           ; SP,*
003664  104414                          TRAP    14
003666  016700  000000G                 MOV     L$UNIT,R0                       ;                                       2633
003672  104451                          TRAP    51
003674  104444                          TRAP    44
003676  062706  000006                  ADD     #6,SP                           ;                                       2630
003702  062706  000032          2$:     ADD     #32,SP                          ;                                       2566
003706  012601                          MOV     (SP)+,R1
003710  000207                          RTS     PC

; Routine Size:  181 words
; Maximum stack depth per invocation:  17 words
```

```
;        2640  routine WRT_TRANSFER (SIZE, DST, SRC) : novalue =
;        2641      begin
;        2642
;        2643  !++
;        2644  ! Functional Description:
;        2645  !       This global routine will perform a write transfer
;        2646  !       at the designated buffer and sector address.
;        2647  !
;        2648  ! Formal Parameters:
;        2649  !       SIZE:
;        2650  !               Stores the size of the transfer
;        2651  !       BUFFER:
;        2652  !               Write buffer address where data is
;        2653  !               coming from or going to.
;        2654  !       SECTOR:
;        2655  !               ML-11 desired sector address where data
;        2656  !               is coming from or going to.
;        2657  !
;        2658  ! Implicit Inputs:
;        2659  !       none
;        2660  !
;        2661  ! Implicit Outputs:
;        2662  !       none
;        2663  !
;        2664  ! Completion codes:
;        2665  !       none
;        2666  !
;        2667  ! Side Effects:
;        2668  !       none
;        2669  !--
;        2670
;        2671      CLR_MBUS;                                   !Clear the mass bus
;        2672      WRT_RH (MLCS2, DRV_SEL, .ML_DUT);           !Select the divice under test
;        2673      WRT_RH (MLWC, WC_REG, .SIZE);               !Load the transfer size
;        2674      WRT_RH (MLDA, DA_REG, .DST);                !Load the destination address
;        2675      WRT_RH (MLBA, BA_REG, .SRC);                !Load the source address
;        2676      WRT_RH (MLCS1, FUNC, FUNC_5);               !Load a write function with go set
;        2677
;        2678      if .ML_ADDR [MLCS1, SC]                     !Does the transfer cause a sc error
;        2679      then
;        2680          begin
;        2681          ERRSF (ERR_13, UNX_DRV_ERR_MSG, DUMPER);        !Report the unexpected error
;        2682          PRINTB (ONE_MSG, WRT_XFER_MSG);         !Print which function caused the error
;        2683          DODU (.L$UNIT);                         !Drop this unit
;        2684          DOCLN;                                  !Jump to the clean up code
;        2685          end;
;        2686
;        2687      do                                          !Delay for transfer to complete
;        2688          0
;        2689      until .ML_ADDR [MLDS, DRY];
;        2690
;        2691      if .ML_ADDR [MLCS1, SC]                     !Did the transfer cause a sc error
```

```
;          2692       then
;          2693           begin
;          2694           ERRSF (ERR_13, UNX_DRV_ERR_MSG, DUMPER);        !Report the unexpected error
;          2695           PRINTP (ONE_MSG, WRT_XFER_MSG);                 !Report which function caused the error
;          2696           DODU (.L$UNIT);                                 !Drop this unit
;          2697           DOCLN;                                          !Jump to the clean up code
;          2698           end;
;          2699
;          2700       end;


                                        .SBTTL  WRT.TRANSFER ROUTINE DECLARATIONS
003712  010146                  WRT.TRANSFER:
                                        MOV     R1,-(SP)                        ;                                       2640
003714  162706  000024                  SUB     #24,SP
003720  016700  040374'                 MOV     ML.ADDR,R0                      ;                                       2641
003724  016066  000010  000022          MOV     10(R0),22(SP)                   ; *,ML.REG
003732  016601  000022                  MOV     22(SP),R1                       ; ML.REG,MLREG
003736  152701  000040                  BISB    #40,R1                          ; *,MLREG
003742  016700  040374'                 MOV     ML.ADDR,R0
003746  010160  000010                  MOV     R1,10(R0)                       ; MLREG,*
003752  016700  040374'                 MOV     ML.ADDR,R0
003756  016066  000010  000020          MOV     10(R0),20(SP)                   ; *,ML.REG
003764  016601  000020                  MOV     20(SP),R1                       ; ML.REG,MLREG
003770  016700  040400'                 MOV     ML.DUT,R0
003774  042700  177770                  BIC     #177770,R0
004000  142701  000007                  BICB    #7,R1                           ; *,MLREG
004004  050001                          BIS     R0,R1                           ; *,MLREG
004006  016700  040374'                 MOV     ML.ADDR,R0
004012  010160  000010                  MOV     R1,10(R0)                       ; MLREG,*
004016  016700  040374'                 MOV     ML.ADDR,R0                      ;                                       2672
004022  016066  000010  000016          MOV     10(R0),16(SP)                   ; *,ML.REG
004030  016601  000016                  MOV     16(SP),R1                       ; ML.REG,MLREG
004034  016700  040400'                 MOV     ML.DUT,R0
004040  042700  177770                  BIC     #177770,R0
004044  142701  000007                  BICB    #7,R1                           ; *,MLREG
004050  050001                          BIS     R0,R1                           ; *,MLREG
004052  016700  040374'                 MOV     ML.ADDR,R0
004056  010160  000010                  MOV     R1,10(R0)                       ; MLREG,*
004062  016700  040374'                 MOV     ML.ADDR,R0                      ;                                       2673
004066  016066  000002  000014          MOV     2(R0),14(SP)                    ; *,ML.REG
004074  016601  000034                  MOV     34(SP),R1                       ; SIZE,MLREG
004100  016700  040374'                 MOV     ML.ADDR,R0
004104  010160  000002                  MOV     R1,2(R0)                        ; MLREG,*
004110  016700  040374'                 MOV     ML.ADDR,R0                      ;                                       2674
004114  016066  000006  000012          MOV     6(R0),12(SP)                    ; *,ML.REG
004122  016601  000032                  MOV     32(SP),R1                       ; DST,MLREG
004126  016700  040374'                 MOV     ML.ADDR,R0
004132  010160  000006                  MOV     R1,6(R0)                        ; MLREG,*
004136  016700  040374'                 MOV     ML.ADDR,R0                      ;                                       2675
004142  016066  000004  000010          MOV     4(R0),10(SP)                    ; *,ML.REG
004150  016601  000030                  MOV     30(SP),R1                       ; SRC,MLREG
```

```
004154  016700  040374'                    MOV     ML.ADDR,R0
004160  010160  000004                      MOV     R1,4(R0)                ; MLREG,*
004164  017766  040374'  000006            MOV     @ML.ADDR,6(SP)          ; *,ML.REG                      2676
004172  016600  000006                      MOV     6(SP),R0                ; ML.REG,MLREG
004176  142700  000077                      BICB    #77,R0                  ; *,MLREG
004202  152700  000061                      BISB    #61,R0                  ; *,MLREG
004206  010077  040374'                    MOV     R0,@ML.ADDR             ; MLREG,*
004212  017766  040374'  000004            MOV     @ML.ADDR,4(SP)          ; *,ML.REG                      2678
004220  100022                            BPL     1$
004222  104454                            TRAP    54                      ;                               2681
004224  000015                            .WORD   15
004226  002310'                           .WORD   UNX.DRV.ERR.MSG
004230  000000'                           .WORD   DUMPER
004232  012746  002422'                    MOV     #WRT.XFER.MSG,-(SP)     ;                               2682
004236  012746  002556'                    MOV     #ONE.MSG,-(SP)
004242  012746  000002                      MOV     #2,-(SP)
004246  010600                            MOV     SP,R0                   ; SP,*
004250  104414                            TRAP    14
004252  016700  000000G                    MOV     L$UNIT,R0               ;                               2683
004256  104451                            TRAP    51
004260  104444                            TRAP    44
004262  062706  000006                      ADD     #6,SP                   ;                               2680
004266  016700  040374'          1$:       MOV     ML.ADDR,R0              ;                               2689
004272  016066  000012  000002            MOV     12(R0),2(SP)            ; *,ML.REG
004300  105766  000002                      TSTB    2(SP)                   ; ML.REG
004304  100370                            BPL     1$
004306  017716  040374'                    MOV     @ML.ADDR,(SP)           ; *,ML.REG                      2691
004312  100022                            BPL     2$
004314  104454                            TRAP    54                      ;                               2694
004316  000015                            .WORD   15
004320  002310'                           .WORD   UNX.DRV.ERR.MSG
004322  000000'                           .WORD   DUMPER
004324  012746  002422'                    MOV     #WRT.XFER.MSG,-(SP)     ;                               2695
004330  012746  002556'                    MOV     #ONE.MSG,-(SP)
004334  012746  000002                      MOV     #2,-(SP)
004340  010600                            MOV     SP,R0                   ; SP,*
004342  104414                            TRAP    14
004344  016700  000000G                    MOV     L$UNIT,R0               ;                               2696
004350  104451                            TRAP    51
004352  104444                            TRAP    44
004354  062706  000006                      ADD     #6,SP                   ;                               2693
004360  062706  000024          2$:       ADD     #24,SP                  ;                               2640
004364  012601                            MOV     (SP)+,R1
004366  000207                            RTS     PC

; Routine Size:  151 words
; Maximum stack depth per invocation:  14 words
```

```
;          2701  routine RD_TRANSFER (SIZE, DST, SRC) : novalue =
;          2702      begin
;          2703
;          2704  !++
;          2705  ! Functional Description:
;          2706  !       This global routine performs a read transfer at the
;          2707  !       designated buffer and source addresses.
;          2708  !
;          2709  ! Formal Parameters:
;          2710  !       SIZE:
;          2711  !               Stores the size of the transfer
;          2712  !       BUFFER:
;          2713  !               Write buffer address where data is
;          2714  !               coming from or going to.
;          2715  !       SECTOR:
;          2716  !               ML-11 desired sector address where data
;          2717  !               is coming from or going to.
;          2718  !
;          2719  ! Implicit Inputs:
;          2720  !       none
;          2721  !
;          2722  ! Implicit Outputs:
;          2723  !       none
;          2724  !
;          2725  ! Completion codes:
;          2726  !       none
;          2727  !
;          2728  ! Side Effects:
;          2729  !       none
;          2730  !--
;          2731
;          2732      CLR_MBUS;                                   !Clear the mass bus
;          2733      ECC_DIS (ENABE);                            !Disable error correction
;          2734      WRT_RH (MLCS2, DRV_SEL, .ML_DUT);           !Select the divice under test
;          2735      WRT_RH (MLWC, WC_REG, .SIZE);               !Load the transfer size
;          2736      WRT_RH (MLDA, DA_REG, .SRC);                !Load the source address
;          2737      WRT_RH (MLBA, BA_REG, .DST);                !Load the destination address
;          2738      WRT_RH (MLCS1, FUNC, FUNC_6);               !Load a read function with go set
;          2739
;          2740      if .ML_ADDR [MLCS1, SC]                     !Does this function cause a sc error
;          2741      then
;          2742          begin
;          2743          ERRSF (ERR_13, UNX_DRV_ERR_MSG, DUMPER);        !Report the unexpected error
;          2744          PRINTB (ONE_MSG, RD_XFER_MSG);          !Report which function cause the error
;          2745          DODU (.L$UNIT);                         !Drop this unit
;          2746          DOCLN;                                  !Jump to the clean up code
;          2747          end;
;          2748
;          2749      do                                          !Delay for the transfer to complete
;          2750          0
;          2751      until .ML_ADDR [MLDS, DRY];
;          2752
```

```
;          2753      if .ML_ADDR [MLCS1, SC]                    !Did the transfer cause a sc error
;          2754      then
;          2755          begin
;          2756          ERRSF (ERR_13, UNX_DRV_ERR_MSG, DUMPER);        !Report the unexpected error
;          2757          PRINTB (ONE_MSG, RD_XFER_MSG);          !Print which function caused the error
;          2758          DODU (.L$UNIT);                         !Drop this unit
;          2759          DOCLN;                                  !Jump to the clean up code
;          2760          end;
;          2761
;          2762      end;


                                        .SBTTL  RD.TRANSFER ROUTINE DECLARATIONS
004370  010146                  RD.TRANSFER:
                                        MOV     R1,-(SP)                ;                                       2701
004372  162706  000026                  SUB     #26,SP
004376  016700  040374'                 MOV     ML.ADDR,R0              ;                                       2702
004402  016066  000010  000024          MOV     10(R0),24(SP)           ; *,ML.REG
004410  016601  000024                  MOV     24(SP),R1               ; ML.REG,MLREG
004414  152701  000040                  BISB    #40,R1                  ; *,MLREG
004420  016700  040374'                 MOV     ML.ADDR,R0
004424  010160  000010                  MOV     R1,10(R0)               ; MLREG,*
004430  016700  040374'                 MOV     ML.ADDR,R0
004434  016066  000010  000022          MOV     10(R0),22(SP)           ; *,ML.REG
004442  016601  000022                  MOV     22(SP),R1               ; ML.REG,MLREG
004446  016700  040400'                 MOV     ML.DUT,R0
004452  042700  177770                  BIC     #177770,R0
004456  142701  000007                  BICB    #7,R1                   ; *,MLREG
004462  050001                          BIS     R0,R1                   ; *,MLREG
004464  016700  040374'                 MOV     ML.ADDR,R0
004470  010160  000010                  MOV     R1,10(R0)               ; MLREG,*
004474  016700  040374'                 MOV     ML.ADDR,R0              ;                                       2733
004500  016066  000024  000020          MOV     24(R0),20(SP)           ; *,ML.REG
004506  016601  000020                  MOV     20(SP),R1               ; ML.REG,MLREG
004512  152701  000002                  BISB    #2,R1                   ; *,MLREG
004516  016700  040374'                 MOV     ML.ADDR,R0
004522  010160  000024                  MOV     R1,24(R0)               ; MLREG,*
004526  016700  040374'                 MOV     ML.ADDR,R0              ;                                       2734
004532  016066  000010  000016          MOV     10(R0),16(SP)           ; *,ML.REG
004540  016601  000016                  MOV     16(SP),R1               ; ML.REG,MLREG
004544  016700  040400'                 MOV     ML.DUT,R0
004550  042700  177770                  BIC     #177770,R0
004554  142701  000007                  BICB    #7,R1                   ; *,MLREG
004560  050001                          BIS     R0,R1                   ; *,MLREG
004562  016700  040374'                 MOV     ML.ADDR,R0
004566  010160  000010                  MOV     R1,10(R0)               ; MLREG,*
004572  016700  040374'                 MOV     ML.ADDR,R0              ;                                       2735
004576  016066  000002  000014          MOV     2(R0),14(SP)            ; *,ML.REG
004604  016601  000036                  MOV     36(SP),R1               ; SIZE,MLREG
004610  016700  040374'                 MOV     ML.ADDR,R0
004614  010160  000002                  MOV     R1,2(R0)                ; MLREG,*
004620  016700  040374'                 MOV     ML.ADDR,R0              ;                                       2736
```

```
004624  016066  000006  000012          MOV     6(R0),12(SP)                    ; *,ML.REG
004632  016601  000032                  MOV     32(SP),R1                       ; SRC,MLREG
004636  016700  040374'                 MOV     ML.ADDR,R0
004642  010160  000006                  MOV     R1,6(R0)                        ; MLREG,*
004646  016700  040374'                 MOV     ML.ADDR,R0                      ;                                       2737
004652  016066  000004  000010          MOV     4(R0),10(SP)                    ; *,ML.REG
004660  016601  000034                  MOV     34(SP),R1                       ; DST,MLREG
004664  016700  040374'                 MOV     ML.ADDR,R0
004670  010160  000004                  MOV     R1,4(R0)                        ; MLREG,*
004674  017766  040374'  000006        MOV     @ML.ADDR,6(SP)                  ; *,ML.REG                              2738
004702  016600  000006                  MOV     6(SP),R0                        ; ML.REG,MLREG
004706  142700  000077                  BICB    #77,R0                          ; *,MLREG
004712  152700  000071                  BISB    #71,R0                          ; *,MLREG
004716  010077  040374'                 MOV     R0,@ML.ADDR                     ; MLREG,*
004722  017766  040374'  000004        MOV     @ML.ADDR,4(SP)                  ; *,ML.REG                              2740
004730  100022                          BPL     1$
004732  104454                          TRAP    54                              ;                                       2743
004734  000015                          .WORD   15
004736  002310'                         .WORD   UNX.DRV.ERR.MSG
004740  000000'                         .WORD   DUMPER
004742  012746  002352'                 MOV     #RD.XFER.MSG,-(SP)              ;                                       2744
004746  012746  002556'                 MOV     #ONE.MSG,-(SP)
004752  012746  000002                  MOV     #2,-(SP)
004756  010600                          MOV     SP,R0                           ; SP,*
004760  104414                          TRAP    14
004762  016700  000000G                 MOV     L$UNIT,R0                       ;                                       2745
004766  104451                          TRAP    51
004770  104444                          TRAP    44
004772  062706  000006                  ADD     #6,SP                           ;                                       2742
004776  016700  040374'          1$:    MOV     ML.ADDR,R0                      ;                                       2751
005002  016066  000012  000002        MOV     12(R0),2(SP)                    ; *,ML.REG
005010  105766  000002                  TSTB    2(SP)                           ; ML.REG
005014  100370                          BPL     1$
005016  017716  040374'                 MOV     @ML.ADDR,(SP)                   ; *,ML.REG                              2753
005022  100022                          BPL     2$
005024  104454                          TRAP    54                              ;                                       2756
005026  000015                          .WORD   15
005030  002310'                         .WORD   UNX.DRV.ERR.MSG
005032  000000'                         .WORD   DUMPER
005034  012746  002352'                 MOV     #RD.XFER.MSG,-(SP)              ;                                       2757
005040  012746  002556'                 MOV     #ONE.MSG,-(SP)
005044  012746  000002                  MOV     #2,-(SP)
005050  010600                          MOV     SP,R0                           ; SP,*
005052  104414                          TRAP    14
005054  016700  000000G                 MOV     L$UNIT,R0                       ;                                       2758
005060  104451                          TRAP    51
005062  104444                          TRAP    44
005064  062706  000006                  ADD     #6,SP                           ;                                       2755
005070  062706  000026          2$:     ADD     #26,SP                          ;                                       2701
005074  012601                          MOV     (SP)+,R1
005076  000207                          RTS     PC
```

```
; Routine Size:  164 words
; Maximum stack depth per invocation:  15 words
```

```
;          2763  routine I_CHIP_TBL : novalue =
;          2764      begin
;          2765
;          2766  !++
;          2767  ! Functional Description:
;          2768  !       The chip table stores for a tested
;          2769  !       bank all the mos ram chip numbers
;          2770  !       which have data errors within them.
;          2771  !
;          2772  !       This routine when called will init
;          2773  !       this table of all previous bad chip
;          2774  !       entries.
;          2775  !
;          2776  ! Formal Parameters:
;          2777  !       none
;          2778  !
;          2779  ! Implicit Inputs:
;          2780  !       CHIP_TBL
;          2781  !
;          2782  ! Implicit Outputs:
;          2783  !       The chip table is loaded with zeros.
;          2784  !
;          2785  ! Completion codes:
;          2786  !       none
;          2787  !
;          2788  ! Side Effects:
;          2789  !       none
;          2790  !--
;          2791
;          2792      !+
;          2793      ! Loop through all table locations
;          2794      ! and initialize the location to
;          2795      ! zero.
;          2796      !-
;          2797
;          2798      incr TBL_INDEX from 0 to 38 do          !Initialize the bad chip table
;          2799          CHIP_TBL [.TBL_INDEX, ALL] = ZEROES;
;          2800
;          2801      end;


                                    .SBTTL  I.CHIP.TBL ROUTINE DECLARATIONS
005100  010146                  I.CHIP.TBL:
                                    MOV     R1,-(SP)                        ;                       2763
005102  005000                      CLR     R0                              ; TBL.INDEX             2798
005104  010001                  1$: MOV     R0,R1                           ; TBL.INDEX,*           2799
005106  006301                      ASL     R1
005110  005061  030230'              CLR     CHIP.TBL(R1)
005114  005200                      INC     R0                              ; TBL.INDEX             2798
005116  020027  000046              CMP     R0,#46                          ; TBL.INDEX,*
005122  003770                      BLE     1$
005124  012601                      MOV     (SP)+,R1                        ;                       2763
```

```
005126  000207                                  RTS     PC

; Routine Size:  12 words
; Maximum stack depth per invocation:  2 words
```

```
;          2802  routine I_TMP_BLST_TBL : novalue =
;          2803      begin
;          2804
;          2805  !++
;          2806  ! Functional Description:
;          2807  !       There are two tables which stores the
;          2808  !       needed information necessary for prom
;          2809  !       maintenance and they are the 'blast table'
;          2810  !       and the 'tempory blast table'.
;          2811  !
;          2812  !       The tempory blast table holds the bad row,
;          2813  !       column, nibble and chip information until
;          2814  !       it can be transfered into the main blast table.
;          2815  !
;          2816  !       This routine when called will clear this temp
;          2817  !       to all zeros.
;          2818  !
;          2819  ! Formal Parameters:
;          2820  !       none
;          2821  !
;          2822  ! Implicit Inputs:
;          2823  !       TMP_BLST_TBL
;          2824  !
;          2825  ! Implicit Outputs:
;          2826  !       The tempory blast table is loaded
;          2827  !       with zeros
;          2828  !
;          2829  ! Completion codes:
;          2830  !       none
;          2831  !
;          2832  ! Side Effects:
;          2833  !       none
;          2834  !--
;          2835
;          2836      !+
;          2837      ! Index through the temporary blast
;          2838      ! table and initialize its locations
;          2839      ! to all zeros.
;          2840      !-
;          2841
;          2842      incr TBL_INDEX from 0 to 9 do              !Initialize the temporary blast table
;          2843          TMP_BLST_TBL [.TBL_INDEX, FULL_WRD] = ZEROES;
;          2844
;          2845      end;


                                        .SBTTL  I.TMP.BLST.TBL ROUTINE DECLARATIONS
005130  010146                  I.TMP.BLST.TBL:
                                        MOV     R1,-(SP)                        ;                       2802
005132  005000                          CLR     R0                              ; TBL.INDEX             2842
005134  010001                  1$:     MOV     R0,R1                           ; TBL.INDEX,*           2843
005136  006301                          ASL     R1
```

```
005140  005061  030346'                        CLR     TMP.BLST.TBL(R1)
005144  005200                                  INC     R0                      ; TBL.INDEX                     2842
005146  020027  000011                          CMP     R0,#11                  ; TBL.INDEX,*
005152  003770                                  BLE     1$
005154  012601                                  MOV     (SP)+,R1                ;                               2802
005156  000207                                  RTS     PC

; Routine Size:  12 words
; Maximum stack depth per invocation:  2 words
```

```
;         2846  routine I_COL_CNT_TBL : novalue =
;         2847      begin
;         2848
;         2849  !++
;         2850  ! Functional Description:
;         2851  !       The column count table stores the number
;         2852  !       of times a particular column number has
;         2853  !       failed with any failing row number.
;         2854  !
;         2855  !       This routine when called will init this
;         2856  !       table to all zeros.
;         2857  !
;         2858  ! Formal Parameters:
;         2859  !       none
;         2860  !
;         2861  ! Implicit Inputs:
;         2862  !       COL_CNT_TBL
;         2863  !
;         2864  ! Implicit Outputs:
;         2865  !       The column count table is loaded with zeros
;         2866  !
;         2867  ! Completion codes:
;         2868  !       none
;         2869  !
;         2870  ! Side Effects:
;         2871  !       none
;         2872  !--
;         2873
;         2874      !+
;         2875      ! Index through all locations in the
;         2876      ! column count table and initialize
;         2877      ! the locations to all zeros.
;         2878      !-
;         2879
;         2880      incr TBL_INDEX from 0 to 255 do           !Initialize the column count table
;         2881          COL_CNT_TBL [.TBL_INDEX] = ZEROES;
;         2882
;         2883      end;


                                        .SBTTL  I.COL.CNT.TBL ROUTINE DECLARATIONS
005160  005000                  I.COL.CNT.TBL:
                                        CLR     R0                      ; TBL.INDEX                     2880
005162  105060  001002'          1$:     CLRB    COL.CNT.TBL(R0)         ; *(TBL.INDEX)                  2881
005166  005200                          INC     R0                      ; TBL.INDEX                     2880
005170  020027  000377                  CMP     R0,#377                 ; TBL.INDEX,*
005174  003772                          BLE     1$
005176  000207                          RTS     PC                      ;                               2846

; Routine Size:  8 words
; Maximum stack depth per invocation:  0 words
```

```
;         2884  routine I_REM_TBL : novalue =
;         2885      begin
;         2886
;         2887  !++
;         2888  ! Functional Description:
;         2889  !       The ramainder table stores all the
;         2890  !       single cell failing rows and columns
;         2891  !       addresses within a failing MOS chip
;         2892  !       that has not yet been blasted.
;         2893  !
;         2894  !       The copied remainder table is a copy of
;         2895  !       the remainder table and is used in determining
;         2896  !       row addresses for blasting.  The remainder table
;         2897  !       is used in determing column addresses for blasting.
;         2898  !
;         2899  !       This routine when called will init these
;         2900  !       tables to all zeros.
;         2901  !
;         2902  ! Formal Parameters:
;         2903  !       none
;         2904  !
;         2905  ! Implicit Inputs:
;         2906  !       REM_TBL, COPIED_REM_TBL
;         2907  !
;         2908  ! Implicit Outputs:
;         2909  !       Both the remainder table and the copied
;         2910  !       remainder tables are loaded with zeros.
;         2911  !
;         2912  ! Completion codes:
;         2913  !       none
;         2914  !
;         2915  ! Side Effects:
;         2916  !       none
;         2917  !--
;         2918
;         2919      !+
;         2920      ! Index through the remainder table
;         2921      ! and the copied remainder table and
;         2922      ! initialize thier locations to zeros.
;         2923      !-
;         2924
;         2925      incr TBL_INDEX from 0 to 99 do              !Initialize the remainder and copied rem table
;         2926          begin
;         2927          REM_TBL [.TBL_INDEX, RO_CL] = ZEROES;
;         2928          COPIED_REM_TBL [.TBL_INDEX, RO_CL] = ZEROES;
;         2929          end;
;         2930
;         2931      end;

                                        .SBTTL  I.REM.TBL ROUTINE DECLARATIONS
005200  010146                    I.REM.TBL:
```

```
                                        MOV     R1,-(SP)                        ;                               2884
                                        CLR     R0                              ; TBL.INDEX                     2925
005202  005000                  1$:    MOV     R0,R1                           ; TBL.INDEX,*                   2927
005204  010001                          ASL     R1
005206  006301                          CLR     REM.TBL(R1)
005210  005061  001402'                 CLR     COPIED.REM.TBL(R1)              ;                               2928
005214  005061  001712'                 INC     R0                              ; TBL.INDEX                     2925
005220  005200                          CMP     R0,#143                         ; TBL.INDEX,*
005222  020027  000143                  BLE     1$
005226  003766                          MOV     (SP)+,R1                        ;                               2884
005230  012601                          RTS     PC
005232  000207

; Routine Size:  14 words
; Maximum stack depth per invocation:  2 words
```

```
;         2932  routine I_COL_PTR : novalue =
;         2933      begin
;         2934
;         2935  !++
;         2936  ! Functional Description:
;         2937  !       The column pointer table points to locations
;         2938  !       in the column count table which have been
;         2939  !       incremented during the present row number being
;         2940  !       searched for failures greater than 10 in the
;         2941  !       error map.  If this row is determined to be all bad (greater
;         2942  !       than 10 failures) then this row number is entered
;         2943  !       into the temp blast table and the count of
;         2944  !       adjacent failing columns in the column count
;         2945  !       table must be decrimented via this pointer table
;         2946  !       which is pointing to their respective table locations.
;         2947  !
;         2948  !       This routine when called will clear this column
;         2949  !       pointer count table to zero.
;         2950  !
;         2951  ! Formal Parameters:
;         2952  !       none
;         2953  !
;         2954  ! Implicit Inputs:
;         2955  !       COL_PTR
;         2956  !
;         2957  ! Implicit Outputs:
;         2958  !       The column pointer table is loaded
;         2959  !       with zeros.
;         2960  !
;         2961  ! Completion codes:
;         2962  !       none
;         2963  !
;         2964  ! Side Effects:
;         2965  !       none
;         2966  !--
;         2967
;         2968      !+
;         2969      ! Index through the column pointer
;         2970      ! table and initilaize the locations
;         2971      ! to zero.
;         2972      !-
;         2973
;         2974      incr TBL_INDEX from 0 to 9 do               !Initialize the column pointer table
;         2975          COL_PTR [.TBL_INDEX] = ZEROES;
;         2976
;         2977      end;


                                        .SBTTL  I.COL.PTR ROUTINE DECLARATIONS
005234  005000                    I.COL.PTR:
                                        CLR     R0                          ; TBL.INDEX                     2974
005236  105060  002222'            1$:     CLRB    COL.PTR(R0)                 ; *(TBL.INDEX)                  2975
```

```
005242  005200                          INC     R0                      ; TBL.INDEX                             2974
005244  020027  000011                  CMP     R0,#11                  ; TBL.INDEX,*
005250  003772                          BLE     1$
005252  000207                          RTS     PC                      ;                                       2932

; Routine Size:  8 words
; Maximum stack depth per invocation:  0 words
```

```
;          2978  routine I_ERROR_MAP : novalue =
;          2979      begin
;          2980
;          2981  !++
;          2982  ! Functional Description:
;          2983  !       The error map stores by sector and
;          2984  !       row number all of the detected failures
;          2985  !       within a tested mos chip.  From this
;          2986  !       map of the failing chip row or column
;          2987  !       addresses can be chosen for masking the
;          2988  !       error out.
;          2989  !
;          2990  !       This routine when called will init the
;          2991  !       error map to all zeros.
;          2992  !
;          2993  ! Formal Parameters:
;          2994  !       none
;          2995  !
;          2996  ! Implicit Inputs:
;          2997  !       ERROR_MAP
;          2998  !
;          2999  ! Implicit Outputs:
;          3000  !       The error map is loaded with zeros.
;          3001  !
;          3002  ! Completion codes:
;          3003  !       none
;          3004  !
;          3005  ! Side Effects:
;          3006  !       none
;          3007  !--
;          3008
;          3009      !+
;          3010      ! Loop through all the blocks and words
;          3011      ! in the error map structure and load the
;          3012      ! locations up with zeros.
;          3013      !-
;          3014
;          3015      incr TBL_INDEX from 0 to %o'17776' by %o'2' do      !Intialize the error map
;          3016          (ERROR_MAP + .TBL_INDEX) = ZEROES;
;          3017
;          3018      end;


                                        .SBTTL  I.ERROR.MAP ROUTINE DECLARATIONS
005254  005000                  I.ERROR.MAP:
                                CLR     R0                      ; TBL.INDEX                     3015
005256  005060  010230'          1$:     CLR     ERROR.MAP(R0)           ; *(TBL.INDEX)          3016
005262  062700  000002                  ADD     #2,R0                   ; *,TBL.INDEX           3015
005266  020027  017776                  CMP     R0,#17776               ; TBL.INDEX,*
005272  003771                          BLE     1$
005274  000207                          RTS     PC                      ;                       2978
```

```
; Routine Size:  9 words
; Maximum stack depth per invocation:  0 words
```

```
;          3019  routine I_BLAST_TBL : novalue =
;          3020      begin
;          3021
;          3022  !++
;          3023  ! Functional Description:
;          3024  !       The blast table stores for all banks of
;          3025  !       a tested array module all blasting information
;          3026  !       needed to mask out data errors.
;          3027  !
;          3028  !       This routine when called will init this table
;          3029  !       to zeros.
;          3030  !
;          3031  ! Formal Parameters:
;          3032  !       none
;          3033  !
;          3034  ! Implicit Inputs:
;          3035  !       BLAST_TBL
;          3036  !
;          3037  ! Implicit Outputs:
;          3038  !       The blast table is loaded up with zeros.
;          3039  !
;          3040  ! Completion codes:
;          3041  !       none
;          3042  !
;          3043  ! Side Effects:
;          3044  !       none
;          3045  !--
;          3046
;          3047      !+
;          3048      ! Loop through all blocks and words and fill
;          3049      ! the words up with zeros.
;          3050      !-
;          3051
;          3052      incr TBL_INDEX from 0 to %o'7776' by %o'2' do       !Initialize the blast table
;          3053          (BLAST_TBL + .TBL_INDEX) = ZEROES;
;          3054
;          3055      end;


                                        .SBTTL  I.BLAST.TBL ROUTINE DECLARATIONS
005276  005000                  I.BLAST.TBL:
                                CLR     R0                          ; TBL.INDEX                 3052
005300  005060  030372'          1$:     CLR     BLAST.TBL(R0)       ; *(TBL.INDEX)              3053
005304  062700  000002                    ADD     #2,R0               ; *,TBL.INDEX               3052
005310  020027  007776                    CMP     R0,#7776            ; TBL.INDEX,*
005314  003771                          BLE     1$
005316  000207                          RTS     PC                  ;                           3019

; Routine Size:  9 words
; Maximum stack depth per invocation:  0 words
```

```
;          3056  routine L_RANDOM_DATA (TEMP_ARR$BNK_SEL) : novalue =
;          3057      begin
;          3058
;          3059  !++
;          3060  ! Functional Description:
;          3061  !       This global routine will load via mass bus
;          3062  !       write transfers random data to the selected
;          3063  !       array and bank addressed by arr$bnk_sel.
;          3064  !
;          3065  ! Formal Parameters:
;          3066  !       ARR$BNK_SEL:
;          3067  !               This argument stores the  array and
;          3068  !               bank select address.
;          3069  !
;          3070  ! Implicit Inputs:
;          3071  !       WRT_BUF, BNK_NUM_SEC
;          3072  !
;          3073  ! Implicit Outputs:
;          3074  !       none
;          3075  !
;          3076  ! Completion codes:
;          3077  !       none
;          3078  !
;          3079  ! Side Effects:
;          3080  !       none
;          3081  !--
;          3082
;          3083      local
;          3084          RANDOM_INDEX;                       !Write buffer random data index
;          3085
;          3086      RANDOM_INDEX = ZERO;                    !The first transfer starts at write buf word zero
;          3087      GEN_RANDOM_DATA ();                     !Generate a write buf of random data
;          3088
;          3089      !+
;          3090      ! Loop through all sectors in this bank and
;          3091      ! fill the sectors up with random data contained in
;          3092      ! the write buffer.  At each sector transfer
;          3093      ! the write buffer index will start two words
;          3094      ! deeper.
;          3095      !-
;          3096
;          3097      incr SEC_CNT from 0 to .BNK_NUM_SEC do      !Load all sectors in bank with random data
;          3098          begin
;          3099          WRT_TRANSFER (SIZE = -256, DST = .TEMP_ARR$BNK_SEL + .SEC_CNT, SRC = WRT_BUF [.RANDOM_INDEX, WRD]);
;          3100          RANDOM_INDEX = .RANDOM_INDEX + 2;       !Next sector starts 2 word deeper in write buffer
;          3101          end;
;          3102
;          3103      end;


                                        .SBTTL  L.RANDOM.DATA ROUTINE DECLARATIONS
005320  004167  000000G              L.RANDOM.DATA:
```

```
                                        JSR     R1,$SAVE4                       ;                               3056
005324  005003                          CLR     R3                              ; RANDOM.INDEX                  3086
005326  004767  173742                  JSR     PC,GEN.RANDOM.DATA              ;                               3087
005332  016704  040416'                 MOV     BNK.NUM.SEC,R4                  ;                               3097
005336  005001                          CLR     R1                              ; SEC.CNT
005340  000430                          BR      2$
005342  012746  177400          1$:     MOV     #-400,-(SP)                     ;                               3099
005346  011667  040426'                 MOV     (SP),SIZE
005352  010102                          MOV     R1,R2                           ; SEC.CNT,*
005354  066602  000016                  ADD     16(SP),R2                       ; TEMP.ARR$BNK.SE,*
005360  010267  040430'                 MOV     R2,DST
005364  010246                          MOV     R2,-(SP)
005366  010302                          MOV     R3,R2                           ; RANDOM.INDEX,*
005370  006302                          ASL     R2
005372  062702  002234'                 ADD     #WRT.BUF,R2
005376  010267  040432'                 MOV     R2,SRC
005402  010246                          MOV     R2,-(SP)
005404  004767  176302                  JSR     PC,WRT.TRANSFER
005410  062703  000002                  ADD     #2,R3                           ; *,RANDOM.INDEX                3100
005414  062706  000006                  ADD     #6,SP                           ;                               3098
005420  005201                          INC     R1                              ; SEC.CNT                       3097
005422  020104          2$:     CMP     R1,R4                           ; SEC.CNT,*
005424  003746                          BLE     1$
005426  000207                          RTS     PC                              ;                               3056

; Routine Size:  36 words
; Maximum stack depth per invocation:  8 words
```

```
;          3104  routine L_1_0_DATA (TEMP_ARR$BNK_SEL, DATA) : novalue =
;          3105      begin
;          3106
;          3107  !++
;          3108  ! Functional Description:
;          3109  !       This global routine will load the
;          3110  !       selected array and bank with ones
;          3111  !       or zeros data.
;          3112  !
;          3113  ! Formal Parameters:
;          3114  !       ARR$BNK_SEL:
;          3115  !               This argument stores the array and
;          3116  !               bank select address.
;          3117  !
;          3118  ! Implicit Inputs:
;          3119  !       WRT_BUF, BNK_NUM_SEC
;          3120  !
;          3121  ! Implicit Outputs:
;          3122  !       WRT_BUF is filled up with data
;          3123  !
;          3124  ! Completion codes:
;          3125  !       none
;          3126  !
;          3127  ! Side Effects:
;          3128  !       none
;          3129  !--
;          3130
;          3131  !
;          3132  ! VER CZMLCB CHANGED 127 TO 255
;          3133  !
;          3134
;          3135      !+
;          3136      ! Fill the write buffer up with the data
;          3137      ! type contained in the argument 'data'
;          3138      !-
;          3139
;          3140      incr WRD_CNT from 0 to (255 + 511*2) do     !Load the write buf with data
;          3141          WRT_BUF [.WRD_CNT, WRD] = .DATA;
;          3142
;          3143      !+
;          3144      ! Now fill this bank under test up with
;          3145      ! the data just load into the write buffer.
;          3146      !-
;          3147
;          3148      incr SEC_CNT from 0 to .BNK_NUM_SEC do      !Load all sectors in bank with data
;          3149          WRT_TRANSFER (SIZE = -256, DST = .TEMP_ARR$BNK_SEL + .SEC_CNT, SRC = WRT_BUF);
;          3150
;          3151      end;


                                        .SBTTL  L.1.0.DATA ROUTINE DECLARATIONS
005430  004167  000000G                L.1.0.DATA:
```

```
                                                JSR     R1,$SAVE3                       ;                               3104
                                                CLR     R0                              ; WRD.CNT                       3140
005434  005000                          1$:     MOV     R0,R1                           ; WRD.CNT,*                     3141
005436  010001
005440  006301                                  ASL     R1
005442  016661  000012  002234'                 MOV     12(SP),WRT.BUF(R1)              ; DATA,*
005450  005200                                  INC     R0                              ; WRD.CNT                       3140
005452  020027  002375                         CMP     R0,#2375                        ; WRD.CNT,*
005456  003767                                  BLE     1$
005460  016703  040416'                         MOV     BNK.NUM.SEC,R3                  ;                               3148
005464  005002                                  CLR     R2                              ; SEC.CNT
005466  000423                                  BR      3$
005470  012746  177400                  2$:     MOV     #-400,-(SP)                     ;                               3149
005474  011667  040426'                         MOV     (SP),SIZE
005500  010201                                  MOV     R2,R1                           ; SEC.CNT,*
005502  066601  000016                         ADD     16(SP),R1                       ; TEMP.ARR$BNK.SE,*
005506  010167  040430'                         MOV     R1,DST
005512  010146                                  MOV     R1,-(SP)
005514  012746  002234'                         MOV     #WRT.BUF,-(SP)
005520  011667  040432'                         MOV     (SP),SRC
005524  004767  176162                         JSR     PC,WRT.TRANSFER
005530  062706  000006                         ADD     #6,SP
005534  005202                                  INC     R2                              ; SEC.CNT                       3148
005536  020203                          3$:     CMP     R2,R3                           ; SEC.CNT,*
005540  003753                                  BLE     2$
005542  000207                                  RTS     PC                              ;                               3104

; Routine Size:  38 words
; Maximum stack depth per invocation:  7 words
```

```
routine L_CHIP_TBL (FAIL_CHIP, PAT_SEL) : novalue =
    begin

!++
! Functional Description:
!       The bad chip table stores for a bank
!       all the failing mos rams and thier
!       failing data patterns.  The possible
!       data patterns that could fail are:
!           1.  all ones data
!           2.  all zeros data
!           3.  thirteen random data patterns
!
!       This global routine will set the fault flag
!       and failing data pattern flag for each
!       mos ram found to be bad.
!
! Formal Parameters:
!       FAIL_CHIP:
!               Points to the failing chips table position
!       PAT_SEL:
!               Points to the current failing pattern for this
!               chip.
!
! Implicit Inputs:
!       CHIP_TBL
!
! Implicit Outputs:
!       Chip tables bit positions fault and either
!       pat_0 or pat_1 or ran_1 to ran_13
!       are selected and set to a 1.
!
! Completion codes:
!       none
!
! Side Effects:
!       none
!--

    CHIP_TBL [.FAIL_CHIP, FAULT] = SET_FLG;      !Set this chips fault flag

    !+
    ! Index into the chip table at this failing
    ! chip and pattern number and set the bit.
    !-

    case .PAT_SEL from 0 to 14 of                !Select the chips failing data pattern
        set

        [0] :
            CHIP_TBL [.FAIL_CHIP, PAT_0] = SET_FLG;      !Set zeroes data flag
```

```
;        3204          [1] :
;        3205              CHIP_TBL [.FAIL_CHIP, PAT_1] = SET_FLG;     !Set ones flag
;        3206
;        3207          [2] :
;        3208              CHIP_TBL [.FAIL_CHIP, RAN_1] = SET_FLG;     !Set random data 1 flag
;        3209
;        3210          [3] :
;        3211              CHIP_TBL [.FAIL_CHIP, RAN_2] = SET_FLG;     !Set random data 2 flag
;        3212
;        3213          [4] :
;        3214              CHIP_TBL [.FAIL_CHIP, RAN_3] = SET_FLG;     !Set random data 3 flag
;        3215
;        3216          [5] :
;        3217              CHIP_TBL [.FAIL_CHIP, RAN_4] = SET_FLG;     !Set random data 4 flag
;        3218
;        3219          [6] :
;        3220              CHIP_TBL [.FAIL_CHIP, RAN_5] = SET_FLG;     !Set random data 5 flag
;        3221
;        3222          [7] :
;        3223              CHIP_TBL [.FAIL_CHIP, RAN_6] = SET_FLG;     !Set random data 6 flag
;        3224
;        3225          [8] :
;        3226              CHIP_TBL [.FAIL_CHIP, RAN_7] = SET_FLG;     !Set random data 7 flag
;        3227
;        3228          [9] :
;        3229              CHIP_TBL [.FAIL_CHIP, RAN_8] = SET_FLG;     !Set random data 8 flag
;        3230
;        3231          [10] :
;        3232              CHIP_TBL [.FAIL_CHIP, RAN_9] = SET_FLG;     !Set random data 9 flag
;        3233
;        3234          [11] :
;        3235              CHIP_TBL [.FAIL_CHIP, RAN_10] = SET_FLG;    !Set random data 10 flag
;        3236
;        3237          [12] :
;        3238              CHIP_TBL [.FAIL_CHIP, RAN_11] = SET_FLG;    !Set random data 11 flag
;        3239
;        3240          [13] :
;        3241              CHIP_TBL [.FAIL_CHIP, RAN_12] = SET_FLG;    !Set random data 12 flag
;        3242
;        3243          [14] :
;        3244              CHIP_TBL [.FAIL_CHIP, RAN_13] = SET_FLG;    !Set random data 13 flag
;        3245          tes;
;        3246
;        3247      end;


                                        .SBTTL  L.CHIP.TBL ROUTINE DECLARATIONS
005544  010146                  L.CHIP.TBL:
                                        MOV     R1,-(SP)                        ;                       3152
005546  016600  000006                   MOV     6(SP),R0                        ; FAIL.CHIP,*           3191
005552  006300                           ASL     R0
005554  062700  030230'                   ADD     #CHIP.TBL,R0
```

```
005560  052710  100000                  BIS     #100000,(R0)
005564  016601  000004                  MOV     4(SP),R1                ; PAT.SEL,*                     3198
005570  006301                          ASL     R1
005572  066107  005576'                ADD     1$(R1),PC
005576  000036                  1$:     .WORD   2$-1$
005600  000044                          .WORD   3$-1$
005602  000052                          .WORD   4$-1$
005604  000060                          .WORD   5$-1$
005606  000066                          .WORD   6$-1$
005610  000074                          .WORD   7$-1$
005612  000102                          .WORD   8$-1$
005614  000110                          .WORD   9$-1$
005616  000116                          .WORD   10$-1$
005620  000124                          .WORD   11$-1$
005622  000132                          .WORD   12$-1$
005624  000140                          .WORD   13$-1$
005626  000146                          .WORD   14$-1$
005630  000154                          .WORD   15$-1$
005632  000162                          .WORD   16$-1$
005634  052710  040000          2$:     BIS     #40000,(R0)             ;                               3202
005640  000451                          BR      17$                     ;                               3198
005642  052710  020000          3$:     BIS     #20000,(R0)             ;                               3205
005646  000446                          BR      17$                     ;                               3198
005650  052710  010000          4$:     BIS     #10000,(R0)             ;                               3208
005654  000443                          BR      17$                     ;                               3198
005656  052710  004000          5$:     BIS     #4000,(R0)              ;                               3211
005662  000440                          BR      17$                     ;                               3198
005664  052710  002000          6$:     BIS     #2000,(R0)              ;                               3214
005670  000435                          BR      17$                     ;                               3198
005672  052710  001000          7$:     BIS     #1000,(R0)              ;                               3217
005676  000432                          BR      17$                     ;                               3198
005700  052710  000400          8$:     BIS     #400,(R0)               ;                               3220
005704  000427                          BR      17$                     ;                               3198
005706  152710  000200          9$:     BISB    #200,(R0)               ;                               3223
005712  000424                          BR      17$                     ;                               3198
005714  152710  000100          10$:    BISB    #100,(R0)               ;                               3226
005720  000421                          BR      17$                     ;                               3198
005722  152710  000040          11$:    BISB    #40,(R0)                ;                               3229
005726  000416                          BR      17$                     ;                               3198
005730  152710  000020          12$:    BISB    #20,(R0)                ;                               3232
005734  000413                          BR      17$                     ;                               3198
005736  152710  000010          13$:    BISB    #10,(R0)                ;                               3235
005742  000410                          BR      17$                     ;                               3198
005744  152710  000004          14$:    BISB    #4,(R0)                 ;                               3238
005750  000405                          BR      17$                     ;                               3198
005752  152710  000002          15$:    BISB    #2,(R0)                 ;                               3241
005756  000402                          BR      17$                     ;                               3198
005760  152710  000001          16$:    BISB    #1,(R0)                 ;                               3244
005764  012601                  17$:    MOV     (SP)+,R1                ;                               3152
005766  000207                          RTS     PC

; Routine Size:  74 words
```

```
; Maximum stack depth per invocation:  2 words
```

```
;        3248  routine L_FAILING_CHIP (TEMP_ARR$BNK_SEL, BAD_CHIP, LST_PAT) =
;        3249      begin
;        3250
;        3251  !++
;        3252  ! Functional Description:
;        3253  !       The program will first search though a
;        3254  !       selected arrays bank and find all
;        3255  !       newly failing mos rams  and store them
;        3256  !       into the chip table.
;        3257  !
;        3258  !       Once they are found they are then loaded
;        3259  !       with the data pattern which failed them
;        3260  !       and failing row column pair addresses are
;        3261  !       searched for.
;        3262  !
;        3263  !       This global routine will load a failing chip
;        3264  !       with one of the data patterns which failed
;        3265  !       during the mass bus transfers.
;        3266  !
;        3267  !
;        3268  ! Formal Parameters:
;        3269  !       ARR$BNK_SEL:
;        3270  !               This argument stores the array and
;        3271  !               bank selcect address.
;        3272  !       BAD_CHIP:
;        3273  !               This points to a failing chip which is
;        3274  !               currently being pm'ed.
;        3275  !       LST_PAT:
;        3276  !               This is a pointer into the failing pattern
;        3277  !               portion of the table.  The failing pattern
;        3278  !               search starts at the last found failing pattern
;        3279  !               and will return with the next failing pattern
;        3280  !               found.
;        3281  !
;        3282  ! Implicit Inputs:
;        3283  !       CHIP_TBL
;        3284  !
;        3285  ! Implicit Outputs:
;        3286  !       none
;        3287  !
;        3288  ! Completion codes:
;        3289  !       At the completion of this routine
;        3290  !       Pat_sel is returned to the operator
;        3291  !       which points to the table location
;        3292  !       which was last referenced.  This
;        3293  !       enables an N search of this table
;        3294  !       rather than an N squared search.
;        3295  !
;        3296  !       If a chip is found to be bad then
;        3297  !       the failing pat_sel is returned.
;        3298  !       A negitive one is returned if no
;        3299  !       chip are found to be bad.
```

```
3300  !
3301  ! Side Effects:
3302  !       none
3303  !--
3304
3305      !+
3306      ! Increment through the chip table and
3307      ! search for failing patterns for this
3308      ! failing chip number.  If a failing
3309      ! pattern is found then return the index
3310      ! value for that pattern else return a
3311      ! -1 to indicate that no failing patterns
3312      ! exist for this chip.
3313      !-
3314
3315      incr PAT_SEL from .LST_PAT to 14 do          !Search th chip table for a failing pattern
3316          begin
3317
3318          !+
3319          ! Index into the chip table at this
3320          ! pattern selected and see if this
3321          ! pattern is set for this failing chip.
3322          !
3323          ! If a se.ected random pattern bit is not
3324          ! set for a failing chip then generate the
3325          ! random data in the write buffer anyways.
3326          ! This will keep the write buffer filled with
3327          ! random data relitive to the random data failing
3328          ! the mass bus transfers.
3329          !-
3330
3331          case .PAT_SEL from 0 to 14 of            !Select a pattern
3332              set
3333
3334              [0] :
3335
3336                  if .CHIP_TBL [.BAD_CHIP, PAT_0] !Did this pattern fail
3337                  then
3338                      begin
3339                      DM_1_0_LOAD (.TEMP_ARR$BNK_SEL, ZEROES);     !Load chip with zeroes
3340                      CHIP_TBL [.BAD_CHIP, PAT_0] = CLR_FLG;       !Clear this pat flag
3341                      FLG_REG [F_RAND_DATA] = CLR_FLG;     !Clear the random flag
3342                      return .PAT_SEL;                !Return which pattern was selected
3343                      end;
3344
3345              [1] :
3346
3347                  if .CHIP_TBL [.BAD_CHIP, PAT_1] !Did this pattern fail
3348                  then
3349                      begin
3350                      DM_1_0_LOAD (.TEMP_ARR$BNK_SEL, ONES);       !Load chip with ones data
3351                      CHIP_TBL [.BAD_CHIP, PAT_1] = CLR_FLG;       !Clear this pat flag
```

```
3352                    FLG_REG [F_RAND_DATA] = CLR_FLG;    !Clear the random flag
3353                    return .PAT_SEL;                    !Return which pattern was selected
3354                    end;
3355
3356            [2] :
3357
3358                if not .CHIP_TBL [.BAD_CHIP, RAN_1]     !Did this pattern fail
3359                then
3360                    begin
3361                    GEN_RANDOM_DATA ();                 !Gen the random data but dont load it
3362                    end
3363                else
3364                    begin
3365                    DM_RAND_LOAD (.TEMP_ARR$BNK_SEL);   !Load the chip with random data
3366                    CHIP_TBL [.BAD_CHIP, RAN_1] = CLR_FLG;      !Clear this pat flag
3367                    FLG_REG [F_RAND_DATA] = SET_FLG;    !Set the random flag
3368                    return .PAT_SEL;                    !Return which pattern was selected
3369                    end;
3370
3371            [3] :
3372
3373                if not .CHIP_TBL [.BAD_CHIP, RAN_2]     !Did this pattern fail
3374                then
3375                    begin
3376                    GEN_RANDOM_DATA ();                 !Gen the random data but dont load it
3377                    end
3378                else
3379                    begin
3380                    DM_RAND_LOAD (.TEMP_ARR$BNK_SEL);   !Load chip with random data
3381                    CHIP_TBL [.BAD_CHIP, RAN_2] = CLR_FLG;      !Clear this pat flag
3382                    FLG_REG [F_RAND_DATA] = SET_FLG;    !Set the random flag
3383                    return .PAT_SEL;                    !Return which pattern was secected
3384                    end;
3385
3386            [4] :
3387
3388                if not .CHIP_TBL [.BAD_CHIP, RAN_3]     !Did this pattern fail
3389                then
3390                    begin
3391                    GEN_RANDOM_DATA ();                 !Gen the random data but dont load it
3392                    end
3393                else
3394                    begin
3395                    DM_RAND_LOAD (.TEMP_ARR$BNK_SEL);   !Load chip with random data
3396                    CHIP_TBL [.BAD_CHIP, RAN_3] = CLR_FLG;      !Clear this pat flag
3397                    FLG_REG [F_RAND_DATA] = SET_FLG;    !Set the random data flag
3398                    return .PAT_SEL;                    !Return which pattern was secected
3399                    end;
3400
3401            [5] :
3402
3403                if not .CHIP_TBL [.BAD_CHIP, RAN_4]     !Did this pattern fail
```

```
3404                    then
3405                        begin
3406                        GEN_RANDOM_DATA ();          !Gen the random data but dont load it
3407                        end
3408                    else
3409                        begin
3410                        DM_RAND_LOAD (.TEMP_ARR$BNK_SEL);   !Load chip with random data
3411                        CHIP_TBL [.BAD_CHIP, RAN_4] = CLR_FLG;      !Clear the pat flag
3412                        FLG_REG [F_RAND_DATA] = SET_FLG;    !Set the random data flag
3413                        return .PAT_SEL;             !Return which pattern was selected
3414                        end;
3415
3416                [6] :
3417
3418                    if not .CHIP_TBL [.BAD_CHIP, RAN_5]     !Did this pattern falil
3419                    then
3420                        begin
3421                        GEN_RANDOM_DATA ();
3422                        end
3423                    else
3424                        begin
3425                        DM_RAND_LOAD (.TEMP_ARR$BNK_SEL);   !Load chip with random data
3426                        CHIP_TBL [.BAD_CHIP, RAN_5] = CLR_FLG;      !Clear the pat flag
3427                        FLG_REG [F_RAND_DATA] = SET_FLG;    !Set the random data flag
3428                        return .PAT_SEL;             !Return which pattern was selected
3429                        end;
3430
3431                [7] :
3432
3433                    if not .CHIP_TBL [.BAD_CHIP, RAN_6]     !Did this pattern fail
3434                    then
3435                        begin
3436                        GEN_RANDOM_DATA ();          !Gen the random data but dont load it
3437                        end
3438                    else
3439                        begin
3440                        DM_RAND_LOAD (.TEMP_ARR$BNK_SEL);   !Load chip with random data
3441                        CHIP_TBL [.BAD_CHIP, RAN_6] = CLR_FLG;      !Clear the pat flag
3442                        FLG_REG [F_RAND_DATA] = SET_FLG;    !Set the random data flag
3443                        return .PAT_SEL;             !Return which pattern was selected
3444                        end;
3445
3446                [8] :
3447
3448                    if not .CHIP_TBL [.BAD_CHIP, RAN_7]     !Did this pattern fail
3449                    then
3450                        begin
3451                        GEN_RANDOM_DATA ();          !Gen the random data but dont load it
3452                        end
3453                    else
3454                        begin
3455                        DM_RAND_LOAD (.TEMP_ARR$BNK_SEL);   !Load chip with random data
```

```
;        3456                    CHIP_TBL [.BAD_CHIP, RAN_7] = CLR_FLG;      !Clear the pat flag
;        3457                    FLG_REG [F_RAND_DATA] = SET_FLG;    !Set the random data flag
;        3458                    return .PAT_SEL;                    !Return which pattern was selected
;        3459                    end;
;        3460
;        3461                [9] :
;        3462
;        3463                    if not .CHIP_TBL [.BAD_CHIP, RAN_8]     !Did this pattern fail
;        3464                    then
;        3465                        begin
;        3466                        GEN_RANDOM_DATA ();             !Gen the random data but dont load it
;        3467                        end
;        3468                    else
;        3469                        begin
;        3470                        DM_RAND_LOAD (.TEMP_ARR$BNK_SEL);   !Load chip with random data
;        3471                        CHIP_TBL [.BAD_CHIP, RAN_8] = CLR_FLG;      !Clear the pat flag
;        3472                        FLG_REG [F_RAND_DATA] = SET_FLG;    !Set the random data flag
;        3473                        return .PAT_SEL;                    !Return which pattern was selected
;        3474                        end;
;        3475
;        3476                [10] :
;        3477
;        3478                    if not .CHIP_TBL [.BAD_CHIP, RAN_9]     !Did this pattern fail
;        3479                    then
;        3480                        begin
;        3481                        GEN_RANDOM_DATA ();             !Gen the random data but dont load it
;        3482                        end
;        3483                    else
;        3484                        begin
;        3485                        DM_RAND_LOAD (.TEMP_ARR$BNK_SEL);   !Load chip with random data
;        3486                        CHIP_TBL [.BAD_CHIP, RAN_9] = CLR_FLG;      !Clear the pat flag
;        3487                        FLG_REG [F_RAND_DATA] = SET_FLG;    !Set the random data flag
;        3488                        return .PAT_SEL;                    !Return which pattern was selected
;        3489                        end;
;        3490
;        3491                [11] :
;        3492
;        3493                    if not .CHIP_TBL [.BAD_CHIP, RAN_10]    !Did this pattern fail
;        3494                    then
;        3495                        begin
;        3496                        GEN_RANDOM_DATA ();             !Gen the random data but dont load it
;        3497                        end
;        3498                    else
;        3499                        begin
;        3500                        DM_RAND_LOAD (.TEMP_ARR$BNK_SEL);   !Load chip with random data
;        3501                        CHIP_TBL [.BAD_CHIP, RAN_10] = CLR_FLG;     !Clear the pat flag
;        3502                        FLG_REG [F_RAND_DATA] = SET_FLG;    !Set the random data flag
;        3503                        return .PAT_SEL;                    !Return which pattern was selected
;        3504                        end;
;        3505
;        3506                [12] :
;        3507
```

```
;          3508                    if not .CHIP_TBL [.BAD_CHIP, RAN_11]    !Did this pattern fail
;          3509                    then
;          3510                        begin
;          3511                        GEN_RANDOM_DATA ();             !Gen the random data but dont load it
;          3512                        end
;          3513                    else
;          3514                        begin
;          3515                        DM_RAND_LOAD (.TEMP_ARR$BNK_SEL);   !Load chip with random data
;          3516                        CHIP_TBL [.BAD_CHIP, RAN_11] = CLR_FLG;     !Clear the pat flag
;          3517                        FLG_REG [F_RAND_DATA] = SET_FLG;    !Set the random data flag
;          3518                        return .PAT_SEL;                !Return which pattern was selected
;          3519                        end;
;          3520
;          3521                [13] :
;          3522
;          3523                    if not .CHIP_TBL [.BAD_CHIP, RAN_12]    !Did this pattern fail
;          3524                    then
;          3525                        begin
;          3526                        GEN_RANDOM_DATA ();             !Gen the random data but dont load it
;          3527                        end
;          3528                    else
;          3529                        begin
;          3530                        DM_RAND_LOAD (.TEMP_ARR$BNK_SEL);   !Load chip with random data
;          3531                        CHIP_TBL [.BAD_CHIP, RAN_12] = CLR_FLG;     !Clear the pat flag
;          3532                        FLG_REG [F_RAND_DATA] = SET_FLG;    !Set the random data flag
;          3533                        return .PAT_SEL;                !Return which pattern was selected
;          3534                        end;
;          3535
;          3536                [14] :
;          3537
;          3538                    if .CHIP_TBL [.BAD_CHIP, RAN_13]        !Did this pattern fail
;          3539                    then
;          3540                        begin
;          3541                        DM_RAND_LOAD (.TEMP_ARR$BNK_SEL);   !Load chip with random data
;          3542                        CHIP_TBL [.BAD_CHIP, RAN_13] = CLR_FLG;     !Clear the pat flag
;          3543                        FLG_REG [F_RAND_DATA] = SET_FLG;    !Set the random data flag
;          3544                        return .PAT_SEL;                !Return which pattern was selected
;          3545                        end;
;          3546
;          3547                tes;
;          3548
;          3549            end;
;          3550
;          3551        return -1;                                      !Return negitive one if all pattern flags are cleared
;          3552        end;


                                        .SBTTL  L.FAILING.CHIP ROUTINE DECLARATIONS
005770  004167  000000G          L.FAILING.CHIP:
                                        JSR     R1,$SAVE2                       ;                                   3248
005774  016600  000012                  MOV     12(SP),R0                       ; BAD.CHIP,*                        3336
006000  006300                          ASL     R0
```

```
006002  012702  030230'                    MOV     #CHIP.TBL,R2
006006  060002                            ADD     R0,R2
006010  016601  000010                    MOV     10(SP),R1               ; LST.PAT,PAT.SEL                       3315
006014  005301                            DEC     R1                      ; PAT.SEL
006016  000167  000560              1$:   JMP     24$
006022  010100                      2$:   MOV     R1,R0                   ; PAT.SEL,*                             3331
006024  006300                            ASL     R0
006026  066007  006032'                    ADD     3$(R0),PC
006032  000036                      3$:   .WORD   4$-3$
006034  000064                            .WORD   5$-3$
006036  000126                            .WORD   7$-3$
006040  000152                            .WORD   8$-3$
006042  000176                            .WORD   9$-3$
006044  000222                            .WORD   10$-3$
006046  000246                            .WORD   11$-3$
006050  000272                            .WORD   12$-3$
006052  000314                            .WORD   13$-3$
006054  000340                            .WORD   14$-3$
006056  000364                            .WORD   15$-3$
006060  000410                            .WORD   16$-3$
006062  000434                            .WORD   17$-3$
006064  000460                            .WORD   18$-3$
006066  000512                            .WORD   21$-3$
006070  032712  040000              4$:   BIT     #40000,(R2)             ;                                       3336
006074  001750                            BEQ     1$
006076  016646  000014                    MOV     14(SP),-(SP)            ; TEMP.ARR$BNK.SE,*                     3339
006102  005046                            CLR     -(SP)
006104  004767  174460                    JSR     PC,DM.1.0.LOAD
006110  042712  040000                    BIC     #40000,(R2)             ;                                       3340
006114  000413                            BR      6$                      ;                                       3341
006116  032712  020000              5$:   BIT     #20000,(R2)             ;                                       3347
006122  001735                            BEQ     1$
006124  016646  000014                    MOV     14(SP),-(SP)            ; TEMP.ARR$BNK.SE,*                     3350
006130  012746  177777                    MOV     #-1,-(SP)
006134  004767  174430                    JSR     PC,DM.1.0.LOAD
006140  042712  020000                    BIC     #20000,(R2)             ;                                       3351
006144  142767  000040  040422'    6$:   BICB    #40,FLG.REG             ;                                       3352
006152  022626                            CMP     (SP)+,(SP)+             ;                                       3347
006154  000167  000416                    JMP     23$                     ;                                       3349
006160  032712  010000              7$:   BIT     #10000,(R2)             ;                                       3358
006164  001555                            BEQ     19$                     ;                                       3361
006166  016646  000014                    MOV     14(SP),-(SP)            ; TEMP.ARR$BNK.SE,*                     3365
006172  004767  173776                    JSR     PC,DM.RAND.LOAD
006176  042712  010000                    BIC     #10000,(R2)             ;                                       3366
006202  000571                            BR      22$                     ;                                       3367
006204  032712  004000              8$:   BIT     #4000,(R2)              ;                                       3373
006210  001543                            BEQ     19$                     ;                                       3376
006212  016646  000014                    MOV     14(SP),-(SP)            ; TEMP.ARR$BNK.SE,*                     3380
006216  004767  173752                    JSR     PC,DM.RAND.LOAD
006222  042712  004000                    BIC     #4000,(R2)              ;                                       3381
006226  000557                            BR      22$                     ;                                       3382
006230  032712  002000              9$:   BIT     #2000,(R2)              ;                                       3388
```

```
006234  001531                          BEQ     19$                             ;                               3391
006236  016646  000014                  MOV     14(SP),-(SP)                    ; TEMP.ARR$BNK.SE,*             3395
006242  004767  173726                  JSR     PC,DM.RAND.LOAD
006246  042712  002000                  BIC     #2000,(R2)                      ;                               3396
006252  000545                          BR      22$                             ;                               3397
006254  032712  001000          10$:    BIT     #1000,(R2)                      ;                               3403
006260  001517                          BEQ     19$                             ;                               3406
006262  016646  000014                  MOV     14(SP),-(SP)                    ; TEMP.ARR$BNK.SE,*             3410
006266  004767  173702                  JSR     PC,DM.RAND.LOAD
006272  042712  001000                  BIC     #1000,(R2)                      ;                               3411
006276  000533                          BR      22$                             ;                               3412
006300  032712  000400          11$:    BIT     #400,(R2)                       ;                               3418
006304  001505                          BEQ     19$                             ;                               3421
006306  016646  000014                  MOV     14(SP),-(SP)                    ; TEMP.ARR$BNK.SE,*             3425
006312  004767  173656                  JSR     PC,DM.RAND.LOAD
006316  042712  000400                  BIC     #400,(R2)                       ;                               3426
006322  000521                          BR      22$                             ;                               3427
006324  105712          12$:    TSTB    (R2)                            ;                               3433
006326  100074                          BPL     19$                             ;                               3436
006330  016646  000014                  MOV     14(SP),-(SP)                    ; TEMP.ARR$BNK.SE,*             3440
006334  004767  173634                  JSR     PC,DM.RAND.LOAD
006340  142712  000200                  BICB    #200,(R2)                       ;                               3441
006344  000510                          BR      22$                             ;                               3442
006346  132712  000100          13$:    BITB    #100,(R2)                       ;                               3448
006352  001462                          BEQ     19$                             ;                               3451
006354  016646  000014                  MOV     14(SP),-(SP)                    ; TEMP.ARR$BNK.SE,*             3455
006360  004767  173610                  JSR     PC,DM.RAND.LOAD
006364  142712  000100                  BICB    #100,(R2)                       ;                               3456
006370  000476                          BR      22$                             ;                               3457
006372  132712  000040          14$:    BITB    #40,(R2)                        ;                               3463
006376  001450                          BEQ     19$                             ;                               3466
006400  016646  000014                  MOV     14(SP),-(SP)                    ; TEMP.ARR$BNK.SE,*             3470
006404  004767  173564                  JSR     PC,DM.RAND.LOAD
006410  142712  000040                  BICB    #40,(R2)                        ;                               3471
006414  000464                          BR      22$                             ;                               3472
006416  132712  000020          15$:    BITB    #20,(R2)                        ;                               3478
006422  001436                          BEQ     19$                             ;                               3481
006424  016646  000014                  MOV     14(SP),-(SP)                    ; TEMP.ARR$BNK.SE,*             3485
006430  004767  173540                  JSR     PC,DM.RAND.LOAD
006434  142712  000020                  BICB    #20,(R2)                        ;                               3486
006440  000452                          BR      22$                             ;                               3487
006442  132712  000010          16$:    BITB    #10,(R2)                        ;                               3493
006446  001424                          BEQ     19$                             ;                               3496
006450  016646  000014                  MOV     14(SP),-(SP)                    ; TEMP.ARR$BNK.SE,*             3500
006454  004767  173514                  JSR     PC,DM.RAND.LOAD
006460  142712  000010                  BICB    #10,(R2)                        ;                               3501
006464  000440                          BR      22$                             ;                               3502
006466  132712  000004          17$:    BITB    #4,(R2)                         ;                               3508
006472  001412                          BEQ     19$                             ;                               3511
006474  016646  000014                  MOV     14(SP),-(SP)                    ; TEMP.ARR$BNK.SE,*             3515
006500  004767  173470                  JSR     PC,DM.RAND.LOAD
006504  142712  000004                  BICB    #4,(R2)                         ;                               3516
```

```
006510  000426                          BR      22$                             ;                               3517
006512  132712  000002                  18$:    BITB    #2,(R2)                 ;                               3523
006516  001003                          BNE     20$
006520  004767  172550                  19$:    JSR     PC,GEN.RANDOM.DATA      ;                               3526
006524  000426                          BR      24$                             ;                               3523
006526  016646  000014                  20$:    MOV     14(SP),-(SP)            ; TEMP.ARR$BNK.SE,*             3530
006532  004767  173436                          JSR     PC,DM.RAND.LOAD
006536  142712  000002                          BICB    #2,(R2)                 ;                               3531
006542  000411                          BR      22$                             ;                               3532
006544  132712  000001                  21$:    BITB    #1,(R2)                 ;                               3538
006550  001414                          BEQ     24$
006552  016646  000014                          MOV     14(SP),-(SP)            ; TEMP.ARR$BNK.SE,*             3541
006556  004767  173412                          JSR     PC,DM.RAND.LOAD
006562  142712  000001                          BICB    #1,(R2)                 ;                               3542
006566  152767  000040  040422'        22$:    BISB    #40,FLG.REG             ;                               3543
006574  005726                          TST     (SP)+                           ;                               3538
006576  010100                  23$:    MOV     R1,R0                           ; PAT.SEL,*                     3540
006600  000207                          RTS     PC
006602  005201                  24$:    INC     R1                              ; PAT.SEL                       3315
006604  020127  000016                  CMP     R1,#16                          ; PAT.SEL,*
006610  003002                          BGT     25$
006612  000167  177204                  JMP     2$
006616  012700  177777                  25$:    MOV     #-1,R0                  ;                               3249
006622  000207                          RTS     PC                              ;                               3248

; Routine Size:  206 words
; Maximum stack depth per invocation:  5 words
```

```
;        3553  routine L_ERROR_MAP (TEMP_ARR$BNK_SEL, BAD_CHIP) : novalue =
;        3554      begin
;        3555
;        3556  !++
;        3557  ! Functional Description:
;        3558  !       The error map stores by sector all the
;        3559  !       newly failing row addresses within a failing chip.
;        3560  !       The failing column address can be calculated
;        3561  !       by adding the row number to the sector number.
;        3562  !
;        3563  !       This global routine searches all sectors within
;        3564  !       a failing chip looking for bad row addresses.
;        3565  !       If a bad row is detected then that rows
;        3566  !       bit position in the error map at this
;        3567  !       sector is set indicating the bad row.
;        3568  !
;        3569  ! Formal Parameters:
;        3570  !       ARR$BNK_SEL:
;        3571  !               This argument stores the array and bank select address
;        3572  !       BAD_CHIP:
;        3573  !               This argument points to a failing which is presently
;        3574  !               being tested.  Dividing this arg by 4 gives the nibble
;        3575  !               position in the ML-11 word and this arg mod 4 gives the
;        3576  !               chips bit position within its nibble.
;        3577  !
;        3578  ! Implicit Inputs:
;        3579  !       ERROR_MAP, FLG_REG, WRT_BUF
;        3580  !
;        3581  ! Implicit Outputs:
;        3582  !       The error map is loaded with failing row address
;        3583  !       locations within the tested chip sector range.
;        3584  !
;        3585  ! Completion codes:
;        3586  !       none
;        3587  !
;        3588  ! Side Effects:
;        3589  !       none
;        3590  !--
;        3591
;        3592      local
;        3593          WRD_INDEX,                          !Points to a word in the error map
;        3594          BIT_INDEX,                          !Points to a bit in the error map
;        3595          RAND_INDEX,                         !Sector random starting position
;        3596          DIAG_REG_SEL,                       !Select data diag reg where failing chip resides
;        3597          BIT_SEL,                            !Select the failing chip position
;        3598          OFFSET;                     !First selects data diag reg; second selects random data offset
;        3599
;        3600      !
;        3601      ! BAD_CHIP / 16 AND BAD_CHIP MOD 16 WILL
;        3602      ! POINT TO WHICH DIAGNOSTIC REGISTER THE BAD
;        3603      ! CHIP CAME FROM AND THE BITS
;        3604      ! POSITION WITHIN THE REGISTER.
```

```
3605        !
3606        OFFSET = .BAD_CHIP/16;                  !Calculate which data diag reg to read
3607        BIT_SEL = .BAD_CHIP mod 16;             !Calculate failing chips bit position
3608        !
3609        !
3610        ! OFFSET VALUES 2 = MLE2 = 14;
3611        !               0 = MLD1 = 15;
3612        !               1 = MLD2 = 16;
3613        !
3614        ! From the value of offset calculated above
3615        ! it can be determined which data diagnostic
3616        ! register this failing bit comes from.
3617        !
3618        ! Therefore examin the contents of offset and
3619        ! determine which register this bit belongs to.
3620        !
3621
3622        if .OFFSET eql 2                        !Is bad chip divided by 16 equal to 2
3623        then
3624            begin
3625            DIAG_REG_SEL = 14;                  !Select data diag reg mle2
3626            BIT_SEL = .BIT_SEL + 8;             !This registers data bits starts in high byte
3627            end
3628        else
3629            DIAG_REG_SEL = .OFFSET + 15;        !0 + 15 = MLD1; 1 + 15 = MLD2
3630
3631        !
3632        ! When random data is loaded into
3633        ! the drive it is done by starting
3634        ! the next sector write transfer two
3635        ! deeper into the write buffer from
3636        ! the previous transfer.
3637        !
3638        ! The flag register will indicate if the
3639        ! data loaded into this bank is random data
3640        ! or not.  If random data was loaded then
3641        ! this routine must also offset into the write
3642        ! buffer two words deeper at each sector transfer.
3643        ! If random data was not loaded then the offset is
3644        ! is not required.
3645        !
3646
3647        if .FLG_REG [F_RAND_DATA] then OFFSET = 2 else OFFSET = ZERO;       !Offset = 2 if random data
3648
3649        DM_RD_TRANSFER (SIZE = -256, DST = RD_BUF, SRC = .TEMP_ARR$BNK_SEL);
3650        RAND_INDEX = ZERO;                      !The first sectors data starts at word zero
3651
3652        !+
3653        ! Before this routine is called this tested bank
3654        ! is filled with a failing data pattern from this
3655        ! failing chip.  Now the failing chip in this bank
3656        ! is read back to find all the previous and newly
```

```
;          3657      ! rows.
;          3658      !
;          3659      ! Increment through all sectors in this bank and
;          3660      ! compare the write data to the data sitting in
;          3661      ! this chip.  On compare errors determine which
;          3662      ! row(s) are in error and indicate the
;          3663      ! failure(s) by setting the row bit at this sector
;          3664      ! in the error map.
;          3665      !-
;          3666
;          3667      incr SEC_NUM from 0 to .BNK_NUM_SEC do      !Read all sectors in failing chip
;          3668          begin
;          3669          BREAK;
;          3670
;          3671          !+
;          3672          ! Increment through all words in this sector
;          3673          ! and find all failing row(s) and store the failing
;          3674          ! number in the error map at this sector.
;          3675          !-
;          3676
;          3677          incr ROW_NUM from 0 to 127 do           !Read all words in failing chip
;          3678              begin
;          3679              DAT_CLK;                            !Clock data into the data diag registers
;          3680
;          3681              !+
;          3682              ! Compare the write data to the read data
;          3683              ! and if in error then indicate the failure
;          3684              ! by setting this row bit at this sector in
;          3685              ! the error map.
;          3686              !-
;          3687
;          3688              if .WRT_BUF [.RAND_INDEX + .ROW_NUM, .BIT_SEL, ONE, ZERO] neq .ML_ADDR [.DIAG_REG_SEL,
;          3689                      .BIT_SEL, ONE, ZERO]        !Is write buf data eql data diag registers data
;          3690              then
;          3691                  begin
;          3692                  WRD_INDEX = .ROW_NUM/16;        !Calculate error map word of failing row
;          3693                  BIT_INDEX = .ROW_NUM mod 16;    !Calculate error map bit pos of failing row
;          3694                  ERROR_MAP [.SEC_NUM, .WRD_INDEX, .BIT_INDEX, ONE, ZERO] = SET_FLG;
;          3695                  FLG_REG [F_ERR_MAP_ENTERED] = SET_FLG;  !Flag that error map was entered
;          3696                  end;
;          3697
;          3698              end;
;          3699
;          3700          !
;          3701          ! Bump the next sector transfer two words
;          3702          ! deeper in the write buffer if this data
;          3703          ! is random data if not random data then
;          3704          ! add zero.
;          3705          !
;          3706      RAND_INDEX = .RAND_INDEX + .OFFSET;         !The next sector starts 2 words deeper if random data
;          3707      end;
;          3708
```

```
;        3709      CLR_MBUS;                                  !Clear the single step dma mode
;        3710      end;


                                        .SBTTL  L.ERROR.MAP ROUTINE DECLARATIONS
006624  004167  000000G           L.ERROR.MAP:
                                    JSR     R1,$SAVE5                   ;                                   3553
006630  162706  000022                SUB     #22,SP
006634  016646  000040                MOV     40(SP),-(SP)                ; BAD.CHIP,*                        3606
006640  012746  000020                MOV     #20,-(SP)
006644  004767  000000G               JSR     PC,BL$DIV
006650  010004                        MOV     R0,R4                       ; *,OFFSET
006652  016616  000044                MOV     44(SP),(SP)                 ; BAD.CHIP,*                        3607
006656  012746  000020                MOV     #20,-(SP)
006662  004767  000000G               JSR     PC,BL$MOD
006666  010066  000006                MOV     R0,6(SP)                    ; *,BIT.SEL
006672  020427  000002                CMP     R4,#2                       ; OFFSET,*                          3622
006676  001006                        BNE     1$
006700  012702  000016                MOV     #16,R2                      ; *,DIAG.REG.SEL                    3625
006704  062766  000010  000006        ADD     #10,6(SP)                   ; *,BIT.SEL                         3626
006712  000403                        BR      2$                          ;                                   3622
006714  010402              1$:       MOV     R4,R2                       ; OFFSET,DIAG.REG.SEL               3629
006716  062702  000017                ADD     #17,R2                      ; *,DIAG.REG.SEL
006722  032767  000040  040422'  2$:  BIT     #40,FLG.REG                 ;                                   3647
006730  001403                        BEQ     3$
006732  012704  000002                MOV     #2,R4                       ; *,OFFSET
006736  000401                        BR      4$
006740  005004              3$:       CLR     R4                          ; OFFSET
006742  012716  177400      4$:       MOV     #-400,(SP)                  ;                                   3649
006746  011667  040426'               MOV     (SP),SIZE
006752  012746  007230'               MOV     #RD.BUF,-(SP)
006756  011667  040430'               MOV     (SP),DST
006762  016646  000052                MOV     52(SP),-(SP)                ; TEMP.ARR$BNK.SE,*
006766  011667  040432'               MOV     (SP),SRC
006772  004767  172366                JSR     PC,DM.RD.TRANSFER
006776  005066  000014                CLR     14(SP)                      ; RAND.INDEX                        3650
007002  016766  040416'  000016       MOV     BNK.NUM.SEC,16(SP)          ;                                   3667
007010  006302                        ASL     R2                          ;                                   3689
007012  005005                        CLR     R5                          ; SEC.NUM                           3667
007014  000536                        BR      8$
007016  104422              5$:       TRAP    22                          ;                                   3668
007020  005001                        CLR     R1                          ; ROW.NUM                           3677
007022  016703  040374'      6$:       MOV     ML.ADDR,R3                  ;                                   3678
007026  016366  000024  000032        MOV     24(R3),32(SP)               ; *,ML.REG
007034  016600  000032                MOV     32(SP),R0                   ; ML.REG,MLREG
007040  152700  000020                BISB    #20,R0                      ; *,MLREG
007044  016703  040374'               MOV     ML.ADDR,R3
007050  010063  000024                MOV     R0,24(R3)                   ; MLREG,*
007054  010103                        MOV     R1,R3                       ; ROW.NUM,*                         3688
007056  066603  000014                ADD     14(SP),R3                   ; RAND.INDEX,*
007062  006303                        ASL     R3
007064  062703  002234'               ADD     #WRT.BUF,R3
```

```
007070  010346                          MOV     R3,-(SP)
007072  016646  000014                  MOV     14(SP),-(SP)            ; BIT.SEL,*
007076  012746  000001                  MOV     #1,-(SP)
007102  005046                          CLR     -(SP)
007104  004767  000000G                 JSR     PC,BL$GT2
007110  062706  000006                  ADD     #6,SP
007114  010003                          MOV     R0,R3
007116  010200                          MOV     R2,R0                   ;                               3689
007120  066700  040374'                 ADD     ML.ADDR,R0
007124  011066  000032                  MOV     (R0),32(SP)             ; *,ML.REG
007130  012716  000032                  MOV     #32,(SP)
007134  060616                          ADD     SP,(SP)                 ; ML.REG,*
007136  016646  000014                  MOV     14(SP),-(SP)            ; BIT.SEL,*                     3688
007142  012746  000001                  MOV     #1,-(SP)
007146  005046                          CLR     -(SP)
007150  004767  000000G                 JSR     PC,BL$GT2
007154  062706  000010                  ADD     #10,SP
007160  020300                          CMP     R3,R0
007162  001444                          BEQ     7$
007164  010146                          MOV     R1,-(SP)                ; ROW.NUM,*                     3692
007166  012746  000020                  MOV     #20,-(SP)
007172  004767  000000G                 JSR     PC,BL$DIV
007176  010066  000026                  MOV     R0,26(SP)               ; *,WRD.INDEX
007202  010116                          MOV     R1,(SP)                 ; ROW.NUM,*                     3693
007204  012746  000020                  MOV     #20,-(SP)
007210  004767  000000G                 JSR     PC,BL$MOD
007214  010066  000026                  MOV     R0,26(SP)               ; *,BIT.INDEX
007220  010500                          MOV     R5,R0                   ; SEC.NUM,*                     3694
007222  006300                          ASL     R0
007224  006300                          ASL     R0
007226  006300                          ASL     R0
007230  066600  000030                  ADD     30(SP),R0               ; WRD.INDEX,*
007234  006300                          ASL     R0
007236  062700  010230'                 ADD     #ERROR.MAP,R0
007242  010016                          MOV     R0,(SP)
007244  016646  000026                  MOV     26(SP),-(SP)            ; BIT.INDEX,*
007250  012746  000001                  MOV     #1,-(SP)
007254  011646                          MOV     (SP),-(SP)
007256  004767  000000G                 JSR     PC,BL$PU2
007262  152767  000004  040422'         BISB    #4,FLG.REG              ;                               3695
007270  062706  000014                  ADD     #14,SP                  ;                               3691
007274  005201                  7$:     INC     R1                      ; ROW.NUM                       3677
007276  020127  000177                  CMP     R1,#177                 ; ROW.NUM,*
007302  003647                          BLE     6$
007304  060466  000014                  ADD     R4,14(SP)               ; OFFSET,RAND.INDEX             3706
007310  005205                          INC     R5                      ; SEC.NUM                       3667
007312  020566  000016          8$:     CMP     R5,16(SP)               ; SEC.NUM,*
007316  003637                          BLE     5$
007320  016701  040374'                 MOV     ML.ADDR,R1              ;                               3707
007324  016166  000010  000026          MOV     10(R1),26(SP)           ; *,ML.REG
007332  016600  000026                  MOV     26(SP),R0               ; ML.REG,MLREG
007336  152700  000040                  BISB    #40,R0                  ; *,MLREG
```

```
007342  016701  040374'                        MOV     ML.ADDR,R1
007346  010061  000010                         MOV     R0,10(R1)                       ; MLREG,*
007352  016701  040374'                        MOV     ML.ADDR,R1
007356  016166  000010  000024                 MOV     10(R1),24(SP)                   ; *,ML.REG
007364  016600  000024                         MOV     24(SP),R0                       ; ML.REG,MLREG
007370  016705  040400'                        MOV     ML.DUT,R5
007374  042705  177770                         BIC     #177770,R5
007400  142700  000007                         BICB    #7,R0                           ; *,MLREG
007404  050500                                 BIS     R5,R0                           ; *,MLREG
007406  016701  040374'                        MOV     ML.ADDR,R1
007412  010061  000010                         MOV     R0,10(R1)                       ; MLREG,*
007416  062706  000034                         ADD     #34,SP                          ;                                       3553
007422  000207                                 RTS     PC

; Routine Size:  192 words
; Maximum stack depth per invocation:  26 words
```

```
;        3711  routine X_TMP_BLST_TBL (TEMP_ARR$BNK_SEL, ALL_BAD_CNT) : novalue =
;        3712      begin
;        3713
;        3714  !++
;        3715  ! Functional Description:
;        3716  !       The temp blast table stores the nibble number,
;        3717  !       row/column number and a r/c select flag for a
;        3718  !       row/colunm which has been selected for blasting.
;        3719  !
;        3720  !       Once it is determined that this chip is not all
;        3721  !       bad (IE. > 10 all bad row/col) then this temp blast
;        3722  !       table is xfered into the main blast table.
;        3723  !
;        3724  !       This table is needed because once this information
;        3725  !       is xfered into the main blast table there is no way
;        3726  !       of knowing which chip this bad row/col came from.
;        3727  !       therefore making it impossible to delete a row/col
;        3728  !       from the blast table once it has been loaded into it.
;        3729  !
;        3730  ! Formal Parameters:
;        3731  !       ARR$BNK_SEL:
;        3732  !               This argument stores the array and bank select
;        3733  !               address.
;        3734  !       ALL_BAD_CNT:
;        3735  !               This argument tells this global routine how many
;        3736  !               remainder table entries to xfer into the
;        3737  !               the main blast table.
;        3738  !
;        3739  ! Implicit Inputs:
;        3740  !       BLAST_TBL, TMP_BLST_TBL
;        3741  !
;        3742  ! Implicit Outputs:
;        3743  !       The tempory blast table contents are transfered
;        3744  !       to the blast table at this bank, row and nibble
;        3745  !       position.
;        3746  !
;        3747  ! Completion codes:
;        3748  !       none
;        3749  !
;        3750  ! Side Effects:
;        3751  !       none
;        3752  !--
;        3753
;        3754      local
;        3755          BNK_NUM;                                    !Points to failing bank
;        3756
;        3757      BNK_NUM = .TEMP_ARR$BNK_SEL<.BNK_SEL_POS, BNK$SEL_SIZE>;    !Calculate the failing bank
;        3758
;        3759      !+
;        3760      ! Index through the temporary blast table
;        3761      ! and set the appropriate row or column bits
;        3762      ! in the main blast table at this row or
```

```
;          3763      ! columns nibble position.
;          3764      !-
;          3765
;          3766      incr CNT from 0 to .ALL_BAD_CNT do          !Transfer temp blast table to blast table
;          3767          !
;          3768          ! If the row bit is set then set the row
;          3769          ! bit in the main blast table at this row,
;          3770          ! bank and nibble number else this is a
;          3771          ! column number and set the column bit in
;          3772          ! the main blast table at this bank and
;          3773          ! nibble number.
;          3774          !
;          3775
;          3776          if .TMP_BLST_TBL [.CNT, R_C] eql 1      !Is this a failing row
;          3777          then
;          3778              BLAST_TBL [.BNK_NUM, .TMP_BLST_TBL [.CNT, R_C_NO], .TMP_BLST_TBL [.CNT, NIB_NO], ONE, ZERO] =
;          3779              SET_FLG                         !Load the blast table with the failing row no. at this nibble
;          3780          else
;          3781              BLAST_TBL [.BNK_NUM, .TMP_BLST_TBL [.CNT, R_C_NO] + .COL_BASE, .TMP_BLST_TBL [.CNT, NIB_NO],
;          3782                  ONE, ZERO] = SET_FLG;           !Load the blast with the failing column at hiis nibble no.
;          3783
;          3784      end;


                                    .SBTTL  X.TMP.BLST.TBL ROUTINE DECLARATIONS
007424  004167  000000G          X.TMP.BLST.TBL:
                                    JSR     R1,$SAVE3                       ;                               3711
007430  012746  000016                MOV     #16.,-(SP)                      ;                               3757
007434  060616                      ADD     SP,(SP)                         ; TEMP.ARR$BNK.SE,*
007436  016746  040414'              MOV     BNK.SEL.POS,-(SP)
007442  012746  000002              MOV     #2,-(SP)
007446  005046                      CLR     -(SP)
007450  004767  000000G              JSR     PC,BL$GT2
007454  000300                      SWAB    R0                              ;                               3778
007456  105000                      CLRB    R0
007460  006300                      ASL     R0
007462  010003                      MOV     R0,R3
007464  005002                      CLR     R2                              ; CNT                           3766
007466  000450                      BR      4$
007470  010201                1$:   MOV     R2,R1                           ; CNT,*                         3776
007472  006301                      ASL     R1
007474  012700  030346'              MOV     #TMP.BLST.TBL,R0
007500  060100                      ADD     R1,R0
007502  005710                      TST     (R0)
007504  100010                      BPL     2$
007506  011001                      MOV     (R0),R1                         ;                               3778
007510  006201                      ASR     R1
007512  006201                      ASR     R1
007514  006201                      ASR     R1
007516  006201                      ASR     R1
007520  042701  177400              BIC     #177400,R1
007524  000411                      BR      3$
```

```
007526  011001                          2$:     MOV     (R0),R1                         ;                                       3781
007530  006201                                  ASR     R1
007532  006201                                  ASR     R1
007534  006201                                  ASR     R1
007536  006201                                  ASR     R1
007540  042701  177400                          BIC     #177400,R1
007544  066701  040372'                        ADD     COL.BASE,R1
007550  060301                          3$:     ADD     R3,R1                           ;                                       3782
007552  006301                                  ASL     R1
007554  062701  030372'                        ADD     #BLAST.TBL,R1
007560  010146                                  MOV     R1,-(SP)
007562  111046                                  MOVB    (R0),-(SP)
007564  042716  177760                          BIC     #177760,(SP)
007570  012746  000001                          MOV     #1,-(SP)
007574  011646                                  MOV     (SP),-(SP)
007576  004767  000000G                         JSR     PC,BL$PU2
007602  062706  000010                          ADD     #10,SP                          ;                                       3776
007606  005202                                  INC     R2                              ; CNT                                   3766
007610  020266  000022                  4$:     CMP     R2,22(SP)                       ; CNT,ALL.BAD.CNT
007614  003725                                  BLE     1$
007616  062706  000010                          ADD     #10,SP                          ;                                       3712
007622  000207                                  RTS     PC                              ;                                       3711

; Routine Size:  64 words
; Maximum stack depth per invocation:  12 words
```

```
;          3785  routine X_TO_REM_TBL (REM_PTR) =
;          3786      begin
;          3787
;          3788  !++
;          3789  ! Functional Description:
;          3790  !       Once the error map and the column count tables
;          3791  !       have been searched for all bad rows and columns all
;          3792  !       that remains are single cell failures  scattered
;          3793  !       through out the error map.
;          3794  !
;          3795  !       This global routine xfers all the remaining row column
;          3796  !       address pairs of the remaining failures into
;          3797  !       the remainder table where they are interigated
;          3798  !       and the best blast selection is determined.
;          3799  !
;          3800  !       It should be noted here that two copies of the
;          3801  !       remainder table exist.  The 'remainder table' will
;          3802  !       be used to manipulate column addresses (the right
;          3803  !       side of the table) and the 'copied remainder table'
;          3804  !       will be used to manipulate row addresses (the left
;          3805  !       side of the table).
;          3806  !
;          3807  ! Formal Parameters:
;          3808  !       REM_PTR
;          3809  !               This argument will first point to the
;          3810  !               starting table position where the first
;          3811  !               transfered row/column pair is to go.
;          3812  !
;          3813  !
;          3814  ! Implicit Inputs:
;          3815  !       ERROR_MAP, REM_TBL, COPIED_REM_TBL, COL_CNT,
;          3816  !       BNK_NUM_SEC, MAX_CHIP_COL
;          3817  !
;          3818  ! Implicit Outputs:
;          3819  !       The remainder and copied remainder table are
;          3820  !       loaded with the remaining single cell failures
;          3821  !       within this tested chip which have not yet been
;          3822  !       selected for blasting.
;          3823  !
;          3824  ! Completion codes:
;          3825  !       Once the global routine is completed 'rem_ptr'
;          3826  !       is returned to indicate how many row/col
;          3827  !       pairs were xfered into the remainder table.
;          3828  !
;          3829  ! Side Effects:
;          3830  !       none
;          3831  !--
;          3832
;          3833      local
;          3834          WRD_INDEX,                      !Stores the adjacent row addresses word position
;          3835          BIT_INDEX,                      !Stores the adjacent row addresses word bit position
;          3836          ROW_NUM,                        !Stores a columns adjacent row number
```

```
3837            ADJACENT_CNT,                   !Stores the number of adjacent row addresses found
3838            START_SEC,                      !Variable sector starting address
3839            END_SEC;                        !Variable sector ending address
3840
3841        !+
3842        ! Index through the column count table and
3843        ! transfer failing columns adjacent row
3844        ! pair into the remainder table.
3845        !-
3846
3847        incr COL_NUM from 0 to .MAX_CHIP_COL - 1 do         !Search column count table
3848            begin
3849            !
3850            ! If this column count table position is
3851            ! not zero then search the error map for
3852            ! its row pair and transfer them into
3853            ! the remainder table.
3854            !
3855
3856            if .COL_CNT_TBL [.COL_NUM] neq ZERO     !Does this location have a count in it
3857            then
3858                begin
3859                !
3860                ! Determine which type chip we are testing
3861                ! and generate the appropriate run time
3862                ! parameters for this routine.
3863                !
3864
3865                if .COL_NUM gtr 127                 !Is this a 64k chip
3866                then
3867                    begin
3868                    ROW_NUM = .COL_NUM - 128;       !The first column address starts at 128
3869                    START_SEC = 256;                !The first sector starts at 256
3870                    END_SEC = 511;                  !The end sector ends at 511
3871                    end
3872                else
3873                    begin
3874                    START_SEC = ZERO;               !The start sector starts at zero
3875                    ROW_NUM = .COL_NUM;             !At the first sector the column = row
3876
3877                    if .BNK_NUM_SEC eql 127 then END_SEC = 127 else END_SEC = 255;  !Is this a 16k chip
3878
3879                    end;
3880
3881                ADJACENT_CNT = ZERO;                !Clear the adjacent count
3882                !
3883                ! Now search thru the error map and find this columns
3884                ! adjacent failing rows using row_num as a pointer to the
3885                ! adjacent row.
3886                !
3887
3888                incr SEC_NUM from .START_SEC to .END_SEC do         !Search all these sectors
```

```
;        3889                    begin
;        3890                    WRD_INDEX = .ROW_NUM/16;        !Calculate which word this row is in
;        3891
;        3892                    if .ERROR_MAP [.SEC_NUM, .WRD_INDEX, FULL_WRD] neq ZERO !Is this row bit set
;        3893                    then
;        3894                        begin
;        3895                        BIT_INDEX = .ROW_NUM mod 16;        !Calculate the rows word bit position
;        3896
;        3897                        if .ERROR_MAP [.SEC_NUM, .WRD_INDEX, .BIT_INDEX, ONE, ZERO] eql 1
;        3898                        then
;        3899                            begin
;        3900                            ADJACENT_CNT = .ADJACENT_CNT + 1;       !Up the adjacent count
;        3901                            REM_TBL [.REM_PTR, COL] = .COL_NUM;     !Load the rem_tbl with this column no.
;        3902                            COPIED_REM_TBL [.REM_PTR, COL] = .COL_NUM;      !Load the copied table too
;        3903
;        3904                            selectone .SEC_NUM of   !Find which physical row number this is
;        3905                                set
;        3906
;        3907                                [0 to 127, 256 to 383] :    !Are we in quad 0 or 2
;        3908                                    begin
;        3909                                    REM_TBL [.REM_PTR, ROW] = .WRD_INDEX*16 + .BIT_INDEX;
;        3910                                    COPIED_REM_TBL [.REM_PTR, ROW] = .WRD_INDEX*16 + .BIT_INDEX;
;        3911                                    end;
;        3912
;        3913                                [128 to 255, 384 to 511] :  !Are we in quad 1 or 3
;        3914                                    begin
;        3915                                    REM_TBL [.REM_PTR, ROW] = .WRD_INDEX*16 + .BIT_INDEX + 128;
;        3916                                    COPIED_REM_TBL [.REM_PTR, ROW] = .WRD_INDEX*16 + .BIT_INDEX + 128;
;        3917                                    end;
;        3918                                tes;
;        3919
;        3920                            REM_PTR = .REM_PTR + 1; !Up the rem_ptr pointer
;        3921
;        3922                            if .ADJACENT_CNT eql .COL_CNT_TBL [.COL_NUM] then exitloop;
;        3923
;        3924                            end;
;        3925
;        3926                        end;
;        3927
;        3928                    ROW_NUM = (.ROW_NUM - 1) and %o'177';   !Decrement the row count and avoide over run
;        3929                    end;
;        3930
;        3931                end;
;        3932
;        3933            end;
;        3934
;        3935        return .REM_PTR;                            !Return how many row column pairs were xfered
;        3936        end;

                                        .SBTTL  X.TO.REM.TBL ROUTINE DECLARATIONS
007624  004167  000000G                X.TO.REM.TBL:
```

```
                                        JSR     R1,$SAVE5                       ;                                       3785
007630  162706  000014                SUB     #14,SP
007634  016766  040376' 000012        MOV     MAX.CHIP.COL,12(SP)             ;                                       3847
007642  005046                        CLR     -(SP)                           ; COL.NUM
007644  000167  000520                JMP     17$
007650  011604                1$:     MOV     (SP),R4                         ; COL.NUM,*                             3856
007652  105764  001002'              TSTB    COL.CNT.TBL(R4)
007656  001002                        BNE     2$
007660  000167  000502                JMP     16$
007664  021627  000177          2$:   CMP     (SP),#177                       ; COL.NUM,*                             3865
007670  003412                        BLE     3$
007672  011601                        MOV     (SP),R1                         ; COL.NUM,ROW.NUM                       3868
007674  162701  000200                SUB     #200,R1                         ; *,ROW.NUM
007700  012766  000400  000010        MOV     #400,10(SP)                     ; *,START.SEC                           3869
007706  012766  000777  000002        MOV     #777,2(SP)                      ; *,END.SEC                             3870
007714  000416                        BR      5$                              ;                                       3865
007716  005066  000010          3$:   CLR     10(SP)                          ; START.SEC                             3874
007722  011601                        MOV     (SP),R1                         ; COL.NUM,ROW.NUM                       3875
007724  026727  040416' 000177        CMP     BNK.NUM.SEC,#177                ;                                       3877
007732  001004                        BNE     4$
007734  012766  000177  000002        MOV     #177,2(SP)                      ; *,END.SEC
007742  000403                        BR      5$
007744  012766  000377  000002  4$:   MOV     #377,2(SP)                      ; *,END.SEC
007752  005066  000012          5$:   CLR     12(SP)                          ; ADJACENT.CNT                          3881
007756  016604  000010                MOV     10(SP),R4                       ; START.SEC,SEC.NUM                     3888
007762  005304                        DEC     R4                              ; SEC.NUM
007764  000574                        BR      15$
007766  010146                6$:     MOV     R1,-(SP)                        ; ROW.NUM,*                             3890
007770  012746  000020                MOV     #20,-(SP)
007774  004767  000000G               JSR     PC,BL$DIV
010000  010066  000012                MOV     R0,12(SP)                       ; *,WRD.INDEX
010004  010400                        MOV     R4,R0                           ; SEC.NUM,*                             3892
010006  006300                        ASL     R0
010010  006300                        ASL     R0
010012  006300                        ASL     R0
010014  010005                        MOV     R0,R5
010016  066605  000012                ADD     12(SP),R5                       ; WRD.INDEX,*
010022  006305                        ASL     R5
010024  005765  010230'              TST     ERROR.MAP(R5)
010030  001544                        BEQ     14$
010032  010146                        MOV     R1,-(SP)                        ; ROW.NUM,*                             3895
010034  012746  000020                MOV     #20,-(SP)
010040  004767  000000G               JSR     PC,BL$MOD
010044  010066  000014                MOV     R0,14(SP)                       ; *,BIT.INDEX
010050  012746  010230'              MOV     #ERROR.MAP,-(SP)                ;                                       3897
010054  060516                        ADD     R5,(SP)
010056  010046                        MOV     R0,-(SP)                        ; BIT.INDEX,*
010060  012746  000001                MOV     #1,-(SP)
010064  005046                        CLR     -(SP)
010066  004767  000000G               JSR     PC,BL$GT2
010072  062706  000010                ADD     #10,SP
010076  005300                        DEC     R0
```

```
010100  001117                          BNE     13$
010102  005266  000022                  INC     22(SP)                  ; ADJACENT.CNT                  3900
010106  016605  000044                  MOV     44(SP),R5               ; REM.PTR,*                     3901
010112  006305                          ASL     R5
010114  012702  001402'                 MOV     #REM.TBL,R2
010120  060502                          ADD     R5,R2
010122  116612  000010                  MOVB    10(SP),(R2)             ; COL.NUM,*
010126  012703  001712'                 MOV     #COPIED.REM.TBL,R3      ;                               3902
010132  060503                          ADD     R5,R3
010134  116613  000010                  MOVB    10(SP),(R3)             ; COL.NUM,*
010140  005704                          TST     R4                      ; SEC.NUM                       3904
010142  002403                          BLT     7$
010144  020427  000177                  CMP     R4,#177                 ; SEC.NUM,*
010150  003406                          BLE     8$
010152  020427  000400          7$:     CMP     R4,#400                 ; SEC.NUM,*
010156  002420                          BLT     9$
010160  020427  000577                  CMP     R4,#577                 ; SEC.NUM,*
010164  003015                          BGT     9$
010166  016600  000016          8$:     MOV     16(SP),R0               ; WRD.INDEX,*                   3909
010172  006300                          ASL     R0
010174  006300                          ASL     R0
010176  006300                          ASL     R0
010200  006300                          ASL     R0
010202  066600  000014                  ADD     14(SP),R0               ; BIT.INDEX,*
010206  110062  000001                  MOVB    R0,1(R2)
010212  110063  000001                  MOVB    R0,1(R3)                ;                               3910
010216  000433                          BR      12$                     ;                               3904
010220  020427  000200          9$:     CMP     R4,#200                 ; SEC.NUM,*
010224  002403                          BLT     10$
010226  020427  000377                  CMP     R4,#377                 ; SEC.NUM,*
010232  003406                          BLE     11$
010234  020427  000600          10$:    CMP     R4,#600                 ; SEC.NUM,*
010240  002422                          BLT     12$
010242  020427  000777                  CMP     R4,#777                 ; SEC.NUM,*
010246  003017                          BGT     12$
010250  016600  000016          11$:    MOV     16(SP),R0               ; WRD.INDEX,*                   3915
010254  006300                          ASL     R0
010256  006300                          ASL     R0
010260  006300                          ASL     R0
010262  006300                          ASL     R0
010264  066600  000014                  ADD     14(SP),R0               ; BIT.INDEX,*
010270  010005                          MOV     R0,R5
010272  062705  000200                  ADD     #200,R5
010276  110562  000001                  MOVB    R5,1(R2)
010302  110563  000001                  MOVB    R5,1(R3)                ;                               3916
010306  005266  000044          12$:    INC     44(SP)                  ; REM.PTR                       3920
010312  005005                          CLR     R5                      ;                               3922
010314  016603  000010                  MOV     10(SP),R3               ; COL.NUM,*
010320  156305  001002'                 BISB    COL.CNT.TBL(R3),R5
010324  026605  000022                  CMP     22(SP),R5               ; ADJACENT.CNT,*
010330  001003                          BNE     13$
010332  062706  000010                  ADD     #10,SP
```

```
010336  000413                          BR      16$
010340  022626                  13$:    CMP     (SP)+,(SP)+                     ;                               3894
010342  010105                  14$:    MOV     R1,R5                           ; ROW.NUM,*                     3928
010344  005305                          DEC     R5
010346  010501                          MOV     R5,R1                           ; *,ROW.NUM
010350  042701  177600                  BIC     #177600,R1                      ; *,ROW.NUM
010354  022626                          CMP     (SP)+,(SP)+                     ;                               3889
010356  005204                  15$:    INC     R4                              ; SEC.NUM                       3888
010360  020466  000002                  CMP     R4,2(SP)                        ; SEC.NUM,END.SEC
010364  003600                          BLE     6$
010366  005216                  16$:    INC     (SP)                            ; COL.NUM                       3847
010370  021666  000014          17$:    CMP     (SP),14(SP)                     ; COL.NUM,*
010374  002002                          BGE     18$
010376  000167  177246                  JMP     1$
010402  016600  000034          18$:    MOV     34(SP),R0                       ; REM.PTR,*                     3786
010406  062706  000016                  ADD     #16,SP                          ;                               3785
010412  000207                          RTS     PC

; Routine Size:  188 words
; Maximum stack depth per invocation:  21 words
```

```
;        3937  routine COL_PURGE (COL_SEL, QUANTITY, LST_REM_ENTRY) : novalue =
;        3938      begin
;        3939
;        3940  !++
;        3941  ! Functional Description:
;        3942  !       The remainder and copied remainder table are loaded with all the
;        3943  !       the remaining single cell failures scattered
;        3944  !       through out the chip.  These tables are interigated
;        3945  !       for the best choice for blasting and once a row
;        3946  !       column pair is selected for blasting it's occurance
;        3947  !       in these tables must be taken out.
;        3948  !
;        3949  !       This global routine will search the remainder tables
;        3950  !       selected column address for blasting and purges the
;        3951  !       table of its occurance and also its row pair address.
;        3952  !
;        3953  !       Once the selected column address is purged the remainder
;        3954  !       table is copied into the copied remainder table.
;        3955  !
;        3956  ! Formal Parameters:
;        3957  !       COL_SEL:
;        3958  !               This argument points to the column selected
;        3959  !               for blasting and the column to be purged from
;        3960  !               the table.
;        3961  !       QUANTITY:
;        3962  !               Indicates how many occurances of this column
;        3963  !               to expect to find in the table.
;        3964  !       LST_REM_ENTRY:
;        3965  !               Points to the last table entry + 1.
;        3966  !
;        3967  ! Implicit Inputs:
;        3968  !       REM_TBL, COPIED_REM_TBL
;        3969  !
;        3970  ! Implicit Outputs:
;        3971  !       The remainder and copied remainder tables are purged
;        3972  !       of column address selected for blasting.
;        3973  !
;        3974  ! Completion codes:
;        3975  !       none
;        3976  !
;        3977  ! Side Effects:
;        3978  !       none
;        3979  !--
;        3980
;        3981      !+
;        3982      ! If only one row or column pair is left
;        3983      ! in the remainder table then there is
;        3984      ! no need to purge.
;        3985      !-
;        3986
;        3987      if .LST_REM_ENTRY gtr ONE                   !Purge the columns if remainder table entries > 1
;        3988      then
```

```
3989            begin
3990
3991            !+
3992            ! Search through the remainder table looking
3993            ! for the occurance of this selected column for
3994            ! purging. Pur_loc will point to the table
3995            ! location where this column is first found.
3996            !-
3997
3998            incr PUR_LOC from 0 to .LST_REM_ENTRY - 1 do    !Purge the remainder table of this column
3999                begin
4000                !
4001                ! Is the contents of this table entry
4002                ! equal to the selected column for purging.
4003                !
4004
4005                if .REM_TBL [.PUR_LOC, COL] eql .COL_SEL    !Is this column to be purged
4006                then
4007                    begin
4008                    !
4009                    ! If this column is located at the bottom
4010                    ! of the table then don't physically move
4011                    ! the entries around.  When this routine
4012                    ! returns the lst_rem_entry will be adjusted
4013                    ! to effectively mask these columns out of
4014                    ! the table.
4015                    !
4016
4017                    if (.PUR_LOC + .QUANTITY) neq .LST_REM_ENTRY    !Is this column last in table
4018                    then
4019                        begin
4020                        !
4021                        ! The selected columns for purging are not at
4022                        ! the end of the buffer so their table placements
4023                        ! must be shuffled around to purge them out.
4024                        !
4025
4026                        incr PURGE from .PUR_LOC to (.LST_REM_ENTRY - 1) - .QUANTITY do
4027                            REM_TBL [.PURGE, RO_CL] = .REM_TBL [.PURGE + .QUANTITY, RO_CL];
4028
4029                        end;
4030
4031                    ! ver czmlcb moved this incr loop here
4032                    !
4033                    ! Once the remainder table has been purged of
4034                    ! this selected column the copied_rem_table
4035                    ! must get a fresh copy to reflect the new
4036                    ! table arrangement.
4037                    !
4038
4039                    incr COPY from 0 to .LST_REM_ENTRY - .QUANTITY do       !Copy rem tbll to copied tbl
4040                        COPIED_REM_TBL [.COPY, RO_CL] = .REM_TBL [.COPY, RO_CL];
```

```
;          4041
;          4042                    exitloop;                       !Exit the loop when purge is complete
;          4043                    end;
;          4044
;          4045                end;
;          4046
;          4047            end;
;          4048
;          4049        end;


                                        .SBTTL  COL.PURGE ROUTINE DECLARATIONS
010414  004167  000000G          COL.PURGE:
                                        JSR     R1,$SAVE5                       ;                                       3937
010420  005746                          TST     -(SP)
010422  016605  000020                  MOV     20(SP),R5                       ; LST.REM.ENTRY,*                       3987
010426  020527  000001                  CMP     R5,#1
010432  003460                          BLE     9$
010434  005016                          CLR     (SP)                            ; PUR.LOC                               3998
010436  000454                          BR      8$
010440  011601                  1$:     MOV     (SP),R1                         ; PUR.LOC,*                             4005
010442  006301                          ASL     R1
010444  005004                          CLR     R4
010446  156104  001402'                 BISB    REM.TBL(R1),R4
010452  020466  000024                  CMP     R4,24(SP)                       ; *,COL.SEL
010456  001043                          BNE     7$
010460  016603  000022                  MOV     22(SP),R3                       ; QUANTITY,*                            4017
010464  010300                          MOV     R3,R0
010466  061600                          ADD     (SP),R0                         ; PUR.LOC,*
010470  020005                          CMP     R0,R5
010472  001420                          BEQ     4$
010474  010502                          MOV     R5,R2                           ;                                       4026
010476  160302                          SUB     R3,R2
010500  011604                          MOV     (SP),R4                         ; PUR.LOC,PURGE
010502  005304                          DEC     R4                              ; PURGE
010504  000410                          BR      3$
010506  010400                  2$:     MOV     R4,R0                           ; PURGE,*                               4027
010510  006300                          ASL     R0
010512  010301                          MOV     R3,R1
010514  060401                          ADD     R4,R1                           ; PURGE,*
010516  006301                          ASL     R1
010520  016160  001402' 001402'        MOV     REM.TBL(R1),REM.TBL(R0)
010526  005204                  3$:     INC     R4                              ; PURGE                                 4026
010530  020402                          CMP     R4,R2                           ; PURGE,*
010532  002765                          BLT     2$
010534  010502                  4$:     MOV     R5,R2                           ;                                       4039
010536  160302                          SUB     R3,R2
010540  005000                          CLR     R0                              ; COPY
010542  000406                          BR      6$
010544  010001                  5$:     MOV     R0,R1                           ; COPY,*                                4040
010546  006301                          ASL     R1
010550  016161  001402' 001712'        MOV     REM.TBL(R1),COPIED.REM.TBL(R1)
```

```
010556  005200                          INC     R0                      ; COPY                          4039
010560  020002                  6$:     CMP     R0,R2                   ; COPY,*
010562  003770                          BLE     5$
010564  000403                          BR      9$                      ;                               4042
010566  005216                  7$:     INC     (SP)                    ; PUR.LOC                       3998
010570  021605                  8$:     CMP     (SP),R5                 ; PUR.LOC,*
010572  002722                          BLT     1$
010574  005726                  9$:     TST     (SP)+                   ;                               3937
010576  000207                          RTS     PC

; Routine Size:  58 words
; Maximum stack depth per invocation:  7 words
```

```
;        4050  routine ROW_PURGE (ROW_SEL, QUANTITY, LST_REM_ENTRY) : novalue =
;        4051      begin
;        4052
;        4053  !++
;        4054  ! Functional Description:
;        4055  !       This global routine is exactly the same as column
;        4056  !       purge except that the copied remainder table is
;        4057  !       search and purged of row address.
;        4058  !
;        4059  ! Formal Parameters:
;        4060  !       ROW_SEL:
;        4061  !               This argument points to the row to be
;        4062  !               purged from the table.
;        4063  !       QUANTITY:
;        4064  !               Indicates how many occurances of this column
;        4065  !               to expect to find in the table.
;        4066  !       LST_REM_ENTRY:
;        4067  !               Points to the last table entry + 1.
;        4068  !
;        4069  ! Implicit Inputs:
;        4070  !       REM_TBL, COPIED_REM_TBL
;        4071  !
;        4072  ! Implicit Outputs:
;        4073  !       The remainder and copied remainder tables are purged
;        4074  !       of all occurances of row addresses selected for blasting.
;        4075  !
;        4076  ! Completion codes:
;        4077  !       none
;        4078  !
;        4079  ! Side Effects:
;        4080  !       none
;        4081  !--
;        4082
;        4083      !+
;        4084      ! If only one row or column pair is left
;        4085      ! in the copied_rem_tble then there is
;        4086      ! no need to purge.
;        4087      !-
;        4088
;        4089      if .LST_REM_ENTRY gtr ONE                   !Purge the row if copied table entries  > 1
;        4090      then
;        4091          begin
;        4092
;        4093          !+
;        4094          ! Search through the copied_rem_tble looking
;        4095          ! for the occurance of this selected row for
;        4096          ! purging. Pur_loc will point to the table
;        4097          ! location where this row is first found.
;        4098          !-
;        4099
;        4100          incr PUR_LOC from 0 to .LST_REM_ENTRY - 1 do    !Purge the copied rem table
;        4101              begin
```

```
4102                !
4103                ! Is the contents of this table entry
4104                ! equal to the selected row for purging.
4105                !
4106
4107                if .COPIED_REM_TBL [.PUR_LOC, ROW] eql .ROW_SEL     !Is this the row to be purged
4108                then
4109                    begin
4110                    !
4111                    ! If this row  is located at the bottom
4112                    ! of the table then don't physically move
4113                    ! the entries around.  When this routine
4114                    ! returns the lst_rem_entry will be adjusted
4115                    ! to effectively mask these rows  out of
4116                    ! the table.
4117                    !
4118
4119                    if (.PUR_LOC + .QUANTITY) neq .LST_REM_ENTRY    !Is this the last table entry
4120                    then
4121                        begin
4122                        !
4123                        ! The selected rows  for purging are not at
4124                        ! the end of the buffer so their table placements
4125                        ! must be shuffled around to purge them out.
4126                        !
4127
4128                        incr PURGE from .PUR_LOC to (.LST_REM_ENTRY - 1) - .QUANTITY do
4129                            COPIED_REM_TBL [.PURGE, RO_CL] = .COPIED_REM_TBL [.PURGE + .QUANTITY, RO_CL];
4130
4131                        end;
4132
4133                    ! ver czmlcb moved this incr loop here
4134                    !
4135                    ! Once the remainder table has been purged of
4136                    ! this selected row  the copied_rem_table
4137                    ! must get a fresh copy to reflect the new
4138                    ! table arrangement.
4139                    !
4140
4141                    incr COPY from 0 to .LST_REM_ENTRY - .QUANTITY do       !Copy copied to rem
4142                        REM_TBL [.COPY, RO_C[] = .COPIED_REM_TBL [.COPY, RO_CL];
4143
4144                    exitloop;                           !Exit loop when purge is completed
4145                    end;
4146
4147                end;
4148
4149            end;
4150
4151        end;
```

```
                                .SBTTL  ROW.PURGE ROUTINE DECLARATIONS
010600  004167  000000G     ROW.PURGE:
                                JSR     R1,$SAVE5                       ;                               4050
010604  005746                        TST     -(SP)
010606  016605  000020              MOV     20(SP),R5                       ; LST.REM.ENTRY,*               4089
010612  020527  000001              CMP     R5,#1
010616  003460                      BLE     9$
010620  005016                      CLR     (SP)                            ; PUR.LOC                       4100
010622  000454                      BR      8$
010624  011601          1$:         MOV     (SP),R1                         ; PUR.LOC,*                     4107
010626  006301                      ASL     R1
010630  005004                      CLR     R4
010632  156104  001713'            BISB    COPIED.REM.TBL+1(R1),R4
010636  020466  000024              CMP     R4,24(SP)                       ; *,ROW.SEL
010642  001043                      BNE     7$
010644  016603  000022              MOV     22(SP),R3                       ; QUANTITY,*                    4119
010650  010300                      MOV     R3,R0
010652  061600                      ADD     (SP),R0                         ; PUR.LOC,*
010654  020005                      CMP     R0,R5
010656  001420                      BEQ     4$
010660  010502                      MOV     R5,R2                           ;                               4128
010662  160302                      SUB     R3,R2
010664  011604                      MOV     (SP),R4                         ; PUR.LOC,PURGE
010666  005304                      DEC     R4                              ; PURGE
010670  000410                      BR      3$
010672  010400          2$:         MOV     R4,R0                           ; PURGE,*                       4129
010674  006300                      ASL     R0
010676  010301                      MOV     R3,R1
010700  060401                      ADD     R4,R1                           ; PURGE,*
010702  006301                      ASL     R1
010704  016160  001712' 001712'    MOV     COPIED.REM.TBL(R1),COPIED.REM.TBL(R0) ;
010712  005204          3$:         INC     R4                              ; PURGE                         4128
010714  020402                      CMP     R4,R2                           ; PURGE,*
010716  002765                      BLT     2$
010720  010502          4$:         MOV     R5,R2                           ;                               4141
010722  160302                      SUB     R3,R2
010724  005000                      CLR     R0                              ; COPY
010726  000406                      BR      6$
010730  010001          5$:         MOV     R0,R1                           ; COPY,*                        4142
010732  006301                      ASL     R1
010734  016161  001712' 001402'    MOV     COPIED.REM.TBL(R1),REM.TBL(R1)
010742  005200                      INC     R0                              ; COPY                          4141
010744  020002          6$:         CMP     R0,R2                           ; COPY,*
010746  003770                      BLE     5$
010750  000403                      BR      9$                              ;                               4144
010752  005216          7$:         INC     (SP)                            ; PUR.LOC                       4100
010754  021605          8$:         CMP     (SP),R5                         ; PUR.LOC,*
010756  002722                      BLT     1$
010760  005726          9$:         TST     (SP)+                           ;                               4050
010762  000207                      RTS     PC

; Routine Size:  58 words
```

```
; Maximum stack depth per invocation:  7 words
```

```
;        4152  routine COL_SORT (LST_REM_ENTRY) : novalue =
;        4153      begin
;        4154
;        4155  !++
;        4156  ! Functional Description:
;        4157  !       Before the remainder table is interigated for columns
;        4158  !       the column side of the table must be sorted.
;        4159  !
;        4160  !       This global routine will sort the column side of the
;        4161  !       remainder table and the row pair will get moved
;        4162  !       around with the column pair.
;        4163  !
;        4164  ! Formal Parameters:
;        4165  !       LST_REM_ENTRY:
;        4166  !               Points to the last remainder table entry
;        4167  !
;        4168  ! Implicit Inputs:
;        4169  !       REM_TBL
;        4170  !
;        4171  ! Implicit Outputs:
;        4172  !       The remainder tables column addresses are sorted
;        4173  !       in ascending order.
;        4174  !
;        4175  ! Completion codes:
;        4176  !       none
;        4177  !
;        4178  ! Side Effects:
;        4179  !       none
;        4180  !--
;        4181
;        4182      local
;        4183          TEMP;                                   !Temporary storage for bubbled entry
;        4184
;        4185      !
;        4186      ! If there is only one table entry
;        4187      ! remaining then there is no need to
;        4188      ! sort the table.
;        4189      !
;        4190
;        4191      if .LST_REM_ENTRY gtr ONE                   !Sort the table if entries are > 1
;        4192      then
;        4193          begin
;        4194
;        4195          !+
;        4196          ! Index each table entry and bubble up
;        4197          ! table values in ascending order.
;        4198          !-
;        4199
;        4200          incr PASS from 1 to .LST_REM_ENTRY - 1 do       !Bubble sort table until in asending order
;        4201
;        4202              !+
;        4203              ! Bubble up this table entry so its
```

```
;        4204                ! in ascending order.
;        4205                !-
;        4206
;        4207                incr index from 0 to ((.LST_REM_ENTRY - 2) - (.PASS - 1)) do
;        4208                                                    !Bubble up the biggest number
;        4209                    !
;        4210                    ! Is this table entry greater then the next
;        4211                    ! table entry and if so then swap the two
;        4212                    ! entries else its in ascending order.
;        4213                    !
;        4214
;        4215                    if .REM_TBL [.index, COL] gtr .REM_TBL [.index + 1, COL]
;        4216                                                    !Is the next entry > this one
;        4217                    then
;        4218                        begin
;        4219                        TEMP = .REM_TBL [.index, RO_CL];    !Copy the bigger entry
;        4220                        REM_TBL [.index, RO_CL] = .REM_TBL [.index + 1, RO_CL];     !Put the smaller entry here
;        4221                        REM_TBL [.index + 1, RO_CL] = .TEMP;        !Put the bigger entry in next entry
;        4222                        end;
;        4223
;        4224            end;
;        4225
;        4226        end;


                                .SBTTL  COL.SORT ROUTINE DECLARATIONS
010764  004167  000000G   COL.SORT:
                            JSR     R1,$SAVE5                       ;                                   4152
010770  005746              TST     -(SP)
010772  016646  000020      MOV     20(SP),-(SP)                    ; LST.REM.ENTRY,*                   4191
010776  021627  000001      CMP     (SP),#1
011002  003441              BLE     6$
011004  011600              MOV     (SP),R0                         ;                                   4207
011006  162700  000002      SUB     #2,R0
011012  005003              CLR     R3                              ; PASS                              4200
011014  000431              BR      5$
011016  010005      1$:     MOV     R0,R5                           ;                                   4207
011020  160305              SUB     R3,R5                           ; PASS,*
011022  010501              MOV     R5,R1
011024  005201              INC     R1
011026  005004              CLR     R4                              ; INDEX
011030  000421              BR      4$
011032  010405      2$:     MOV     R4,R5                           ; INDEX,*                           4215
011034  006305              ASL     R5
011036  012702  001402'      MOV     #REM.TBL,R2
011042  060502              ADD     R5,R2
011044  010405              MOV     R4,R5                           ; INDEX,*
011046  006305              ASL     R5
011050  062705  001404'      ADD     #REM.TBL+2,R5
011054  121215              CMPB    (R2),(R5)
011056  101405              BLOS    3$
011060  011266  000002      MOV     (R2),2(SP)                      ; *,TEMP                            4219
```

```
011064  011512                          MOV     (R5),(R2)                       ;                                       4220
011066  016615  000002                  MOV     2(SP),(R5)                      ; TEMP,*                                4221
011072  005204                  3$:     INC     R4                              ; INDEX                                 4207
011074  020401                  4$:     CMP     R4,R1                           ; INDEX,*
011076  003755                          BLE     2$
011100  005203                  5$:     INC     R3                              ; PASS                                  4200
011102  020316                          CMP     R3,(SP)                         ; PASS,*
011104  002744                          BLT     1$
011106  022626                  6$:     CMP     (SP)+,(SP)+                     ;                                       4152
011110  000207                          RTS     PC

; Routine Size:  43 words
; Maximum stack depth per invocation:  8 words
```

```
;        4227  routine ROW_SORT (LST_REM_ENTRY) : novalue =
;        4228      begin
;        4229
;        4230  !++
;        4231  ! Functional Description:
;        4232  !       This global routine serves the same purpose as
;        4233  !       column sort except the copied remainder table
;        4234  !       row addresses are sorted in ascending order.
;        4235  !
;        4236  ! Formal Parameters:
;        4237  !       LST_REM_ENTRY:
;        4238  !               Points to the last remainder table entry
;        4239  !
;        4240  ! Implicit Inputs:
;        4241  !       COPIED_REM_TBL
;        4242  !
;        4243  ! Implicit Outputs:
;        4244  !       The copied remainder tables row addresses are sorted
;        4245  !       in ascending order.
;        4246  !
;        4247  ! Completion codes:
;        4248  !       none
;        4249  !
;        4250  ! Side Effects:
;        4251  !       none
;        4252  !--
;        4253
;        4254      local
;        4255          TEMP;                                   !Tempory storage for bubbled entry
;        4256
;        4257      !
;        4258      ! If there is only one table entry
;        4259      ! remaining then there is no need to
;        4260      ! sort the table.
;        4261      !
;        4262
;        4263      if .LST_REM_ENTRY gtr ONE                   !Sort the table if entries > 1
;        4264      then
;        4265          begin
;        4266
;        4267          !+
;        4268          ! Index each table entry and bubble up
;        4269          ! table values in ascending order.
;        4270          !-
;        4271
;        4272          incr PASS from 1 to .LST_REM_ENTRY - 1 do       !Bubble sort table until in aseending order
;        4273
;        4274              !+
;        4275              ! Bubble up this table entry so its
;        4276              ! in ascending order.
;        4277              !-
;        4278
```

```
;        4279              incr index from 0 to ((.LST_REM_ENTRY - 2) - (.PASS - 1)) do
;        4280                                                  !Bubble up the biggest number
;        4281                  !
;        4282                  ! Is this table entry greater then the next
;        4283                  ! table entry and if so then swap the two
;        4284                  ! entries else its in ascending order.
;        4285                  !
;        4286
;        4287                  if .COPIED_REM_TBL [.index, ROW] gtr .COPIED_REM_TBL [.index + 1, ROW]
;        4288                  then
;        4289                      begin
;        4290                      TEMP = .COPIED_REM_TBL [.index, RO_CL];     !Copy the bigger entry
;        4291                      COPIED_REM_TBL [.index, RO_CL] = .COPIED_REM_TBL [.index + 1, RO_CL];
;        4292                      COPIED_REM_TBL [.index + 1, RO_CL] = .TEMP; !Put the bigger entyr in next entry
;        4293                      end;
;        4294
;        4295              end;
;        4296
;        4297      end;


                                        .SBTTL  ROW.SORT ROUTINE DECLARATIONS
011112  004167  000000G           ROW.SORT:
                                        JSR     R1,$SAVE5                       ;                               4227
011116  005746                          TST     -(SP)
011120  016646  000020                  MOV     20(SP),-(SP)                    ; LST.REM.ENTRY,*               4263
011124  021627  000001                  CMP     (SP),#1
011130  003443                          BLE     6$
011132  011600                          MOV     (SP),R0                         ;                               4279
011134  162700  000002                  SUB     #2,R0
011140  005003                          CLR     R3                              ; PASS                          4272
011142  000433                          BR      5$
011144  010005                  1$:     MOV     R0,R5                           ;                               4279
011146  160305                          SUB     R3,R5                           ; PASS,*
011150  010501                          MOV     R5,R1
011152  005201                          INC     R1
011154  005004                          CLR     R4                              ; INDEX
011156  000423                          BR      4$
011160  010405                  2$:     MOV     R4,R5                           ; INDEX,*                       4287
011162  006305                          ASL     R5
011164  012702  001712'                  MOV     #COPIED.REM.TBL,R2
011170  060502                          ADD     R5,R2
011172  010405                          MOV     R4,R5                           ; INDEX,*
011174  006305                          ASL     R5
011176  062705  001714'                  ADD     #COPIED.REM.TBL+2,R5
011202  126265  000001  000001          CMPB    1(R2),1(R5)
011210  101405                          BLOS    3$
011212  011266  000002                  MOV     (R2),2(SP)                      ; *,TEMP                        4290
011216  011512                          MOV     (R5),(R2)                       ;                               4291
011220  016615  000002                  MOV     2(SP),(R5)                      ; TEMP,*                        4292
011224  005204                  3$:     INC     R4                              ; INDEX                         4279
011226  020401                  4$:     CMP     R4,R1                           ; INDEX,*
```

```
011230  003753                          BLE     2$
011232  005203          5$:             INC     R3                      ; PASS                  4272
011234  020316                          CMP     R3,(SP)                 ; PASS,*
011236  002742                          BLT     1$
011240  022626          6$:             CMP     (SP)+,(SP)+             ;                       4227
011242  000207                          RTS     PC

; Routine Size:  45 words
; Maximum stack depth per invocation:  8 words
```

```
;        4298  routine S_BLAST_TBL (BNK_NUM) =
;        4299      begin
;        4300
;        4301  !++
;        4302  ! Functional Description:
;        4303  !       To determine if a bank has any additional
;        4304  !       failing row or columns the blast table is
;        4305  !       searched for any bits set.
;        4306  !
;        4307  !       If bits are set then this bank must be
;        4308  !       pm'ed.  If no bits are set then this bank
;        4309  !       can be skipped.
;        4310  !
;        4311  !       This global routine searches the blast table at
;        4312  !       the selected banks and sees if any bits are set
;        4313  !       indicating new errors were found in this bank.
;        4314  !
;        4315  ! Formal Parameters:
;        4316  !       BNK_NUM:
;        4317  !               Points to the bank to be searched
;        4318  !
;        4319  ! Implicit Inputs:
;        4320  !       BLAST_TBL, MAX_CHIP_COL
;        4321  !
;        4322  ! Implicit Outputs:
;        4323  !       none
;        4324  !
;        4325  ! Completion codes:
;        4326  !       A true indicator is returned if new errors
;        4327  !       exist and a false indicator is returned if
;        4328  !       no new errors are found.
;        4329  !
;        4330  ! Side Effects:
;        4331  !       none
;        4332  !--
;        4333
;        4334      !+
;        4335      ! Search all row and column blast information
;        4336      ! stored in the blast table for this bank and
;        4337      ! see if any new rows or columns need to be
;        4338      ! blasted.  If new blast info is found then
;        4339      ! return true as the routines value else return
;        4340      ! false indicating that no blasting is to be done
;        4341      ! for this bank.
;        4342      !-
;        4343
;        4344      incr SEARCH from 0 to .MAX_CHIP_COL*2 - 1 do        !Look at all rows and cols for this bank
;        4345          begin
;        4346          !
;        4347          ! Does this table location contain new
;        4348          ! prom blast information?
;        4349          !
```

```
;          4350
;          4351          if .BLAST_TBL [.BNK_NUM, .SEARCH, FULL_WRD] neq ZERO then return TRUE;  !Return tru if any bits set
;          4352
;          4353          end;
;          4354
;          4355      return FALSE;                                   !Return false if no bits are set
;          4356      end;


                                        .SBTTL  S.BLAST.TBL RCUTINE DECLARATIONS
011244  004167  000000G          S.BLAST.TBL:
                                        JSR     R1,$SAVE3                       ;                                 4298
011250  016703  040376'                 MOV     MAX.CHIP.COL,R3                 ;                                 4344
011254  006303                          ASL     R3
011256  016600  000012                  MOV     12(SP),R0                       ; BNK.NUM,*                       4351
011262  000300                          SWAB    R0
011264  105000                          CLRB    R0
011266  006300                          ASL     R0
011270  005001                          CLR     R1                              ; SEARCH                          4344
011272  000412                          BR      3$
011274  010002                  1$:     MOV     R0,R2                           ;                                 4351
011276  060102                          ADD     R1,R2                           ; SEARCH,*
011300  006302                          ASL     R2
011302  005762  030372'                 TST     BLAST.TBL(R2)
011306  001403                          BEQ     2$
011310  012700  000001                  MOV     #1,R0
011314  000207                          RTS     PC
011316  005201                  2$:     INC     R1                              ; SEARCH                          4344
011320  020103                  3$:     CMP     R1,R3                           ; SEARCH,*
011322  002764                          BLT     1$
011324  005000                          CLR     R0                              ;                                 4299
011326  000207                          RTS     PC                              ;                                 4298

; Routine Size:  26 words
; Maximum stack depth per invocation:  4 words
```

```
;        4357  routine S_COLUMNS (MOST_OFTEN, SUM_MOST  LST_REM_ENTRY) =
;        4358      begin
;        4359
;        4360  !++
;        4361  ! Functional Description:
;        4362  !       The remainder table is searched for the
;        4363  !       best possible choice for blasting ie
;        4364  !       whether to select rows or columns  for
;        4365  !       blasting.  This is done by:
;        4366  !
;        4367  !               While searching through the rem_tbl count how
;        4368  !               many different row numbers and column numbers
;        4369  !               there are.
;        4370  !
;        4371  !               In the lower count group (default to rows if
;        4372  !               counts are equal) find which member appears
;        4373  !               most often.
;        4374  !
;        4375  !               The member which appears most often in this
;        4376  !               lower count group will be the one chosen for
;        4377  !               blasting.
;        4378  !
;        4379  !               For example:
;        4380  !                       Suppose that these row column pairs
;        4381  !                       exist in the remainder table.
;        4382  !
;        4383  !                               ROW         COLUMN
;        4384  !                                2            5
;        4385  !                                3            5
;        4386  !                                4            5
;        4387  !                                9            6
;        4388  !                               10            6
;        4389  !                               11            6
;        4390  !                               12            6
;        4391  !                               99            1
;        4392  !                              100            1
;        4393  !                               60            7
;        4394  !                                2           10
;        4395  !
;        4396  !                       From this there are 5 different column numbers
;        4397  !                       and column number 6 appears the most often and
;        4398  !                       it appears 4 times.
;        4399  !
;        4400  !                       Therefore   DIF = 5
;        4401  !                                   MOST_OFTEN = 6
;        4402  !                                   SUM_MOST = 4
;        4403  !
;        4404  ! Formal Parameters:
;        4405  !       MOST_OFTEN:
;        4406  !               Passed by reference and stores the most
;        4407  !               often found column.
;        4408  !       SUM_OFTEN:
```

```
4409  !                Passed by reference and stores the sum
4410  !                of the most often found column.
4411  !        LST_REM_ENTRY:
4412  !                Points to the last remainder entry
4413  !
4414  ! Implicit Inputs:
4415  !        REM_TBL
4416  !
4417  ! Implicit Outputs:
4418  !        MOST_OFTEN AND SUM_MOST which are passed by reference
4419  !        to this routine is returned to the caller by reference
4420  !        with the column number which appears most often and the
4421  !        number of times it appears.
4422  !
4423  ! Completion codes:
4424  !        The number of different column numbers which were found
4425  !        in the remainder table is returned to the caller.
4426  !
4427  ! Side Effects:
4428  !        none
4429  !--
4430
4431      local
4432          CUR_SUM,                                !Stores current sum of most often found column
4433          PRE_SUM,                                !Stores previous sum of most often found column
4434          CUR_MOST,                               !Stores current most often found column
4435          PRE_MOST,                               !Stores previous most often found column
4436          DIF;                                    !Stores how many different columns were found
4437
4438          !
4439          ! Start out the search by defining
4440          ! the current count, previous count
4441          ! and different count to be one.
4442          !
4443          ! Also define the first table entry
4444          ! to be the column number to be selected
4445          ! for blasting.
4446          !
4447          ! This is done in the event that there
4448          ! is only one table entry left to be
4449          ! selected for blasting.
4450          !
4451          CUR_SUM = 1;                            !Current sum starts at one
4452          PRE_SUM = 1;                            !Previous sum starts at onc
4453          PRE_MOST = .REM_TBL [ZERO, COL];        !Previous most often starts at first column in table
4454          DIF = 1;                                !Different column count starts at one
4455
4456          !+
4457          ! If only one table entry is remaining
4458          ! in the table then return with the default
4459          ! values defined above as the routines return
4460          ! values.
```

```
;        4461      !-
;        4462
;        4463      if .LST_REM_ENTRY gtr ONE                    !First table entry is most oftem if table entry < 1
;        4464      then
;        4465          begin
;        4466          COL_SORT (.LST_REM_ENTRY);               !Sort the column side of remainder table
;        4467          !
;        4468          ! Index through the table and search for
;        4469          ! the column most often found and the number
;        4470          ! of time its found. Also search the table
;        4471          ! for the number of different column numbers
;        4472          ! found.
;        4473          !
;        4474
;        4475          incr index from 0 to .LST_REM_ENTRY - 2 do      !Find the sum, most often and different
;        4476              begin
;        4477              !
;        4478              ! Is this table position contents the same as the
;        4479              ! next table contents?
;        4480              !
;        4481
;        4482              if .REM_TBL [.index, COL] eql .REM_TBL [.index + 1, COL]    !Is this the same as the next
;        4483              then
;        4484                  begin
;        4485                  CUR_MOST = .REM_TBL [.index, COL];      !Most often is this entry
;        4486                  CUR_SUM = .CUR_SUM + 1;          !Increment the sum of the most
;        4487                  end
;        4488              else
;        4489                  begin
;        4490                  DIF = .DIF + 1;                  !Increment the different column count
;        4491
;        4492                  if .CUR_SUM gtr .PRE_SUM         !Is the current sum > previous sum
;        4493                  then
;        4494                      begin
;        4495                      PRE_SUM = .CUR_SUM;          !Save the sum of the current most often
;        4496                      PRE_MOST = .CUR_MOST;        !Save the current most often
;        4497                      end;
;        4498
;        4499                  CUR_SUM = 1;                     !Start current sum back to one
;        4500                  end;
;        4501
;        4502              end;
;        4503
;        4504          end;
;        4505
;        4506      !
;        4507      ! This test to see if the last set of
;        4508      ! unique column numbers in the table is > the previous sum
;        4509      !
;        4510
;        4511      if .CUR_SUM gtr .PRE_SUM                     !Is the current sum > the previous sum
;        4512      then
```

```
;        4513          begin
;        4514          PRE_SUM = .CUR_SUM;                 !Save the sum of the current most often
;        4515          PRE_MOST = .CUR_MOST;               !Save the current most often
;        4516          end;
;        4517
;        4518      .MOST_OFTEN = .PRE_MOST;                !Return the most often found column
;        4519      .SUM_MOST = .PRE_SUM;                   !Return the sum of the most often column
;        4520      return .DIF;                            !Return the different number of columns found
;        4521      end;


                                        .SBTTL  S.COLUMNS ROUTINE DECLARATIONS
011330  004167  000000G           S.COLUMNS:
                                        JSR     R1,$SAVE5                       ;                       4357
011334  024646                            CMP     -(SP),-(SP)
011336  012701  000001                    MOV     #1,R1                           ; *,CUR.SUM             4451
011342  010105                            MOV     R1,R5                           ; *,PRE.SUM             4452
011344  005046                            CLR     -(SP)                           ; PRE.MOST              4453
011346  116716  001402'                   MOVB    REM.TBL,(SP)                    ; *,PRE.MOST
011352  012766  000001  000002            MOV     #1,2(SP)                        ; *,DIF                 4454
011360  016604  000024                    MOV     24(SP),R4                       ; LST.REM.ENTRY,*       4463
011364  020427  000001                    CMP     R4,#1
011370  003444                            BLE     6$
011372  010446                            MOV     R4,-(SP)                        ;                       4466
011374  004767  177364                    JSR     PC,COL.SORT
011400  162704  000002                    SUB     #2,R4                           ;                       4475
011404  005002                            CLR     R2                              ; INDEX
011406  000432                            BR      5$
011410  010203                    1$:     MOV     R2,R3                           ; INDEX,*               4482
011412  006303                            ASL     R3
011414  010200                            MOV     R2,R0                           ; INDEX,*
011416  006300                            ASL     R0
011420  126360  001402' 001404'           CMPB    REM.TBL(R3),REM.TBL+2(R0)
011426  001007                            BNE     2$
011430  116366  001402' 000006            MOVB    REM.TBL(R3),6(SP)               ; *,CUR.MOST            4485
011436  105066  000007                    CLRB    7(SP)                           ; CUR.MOST
011442  005201                            INC     R1                              ; CUR.SUM               4486
011444  000412                            BR      4$                              ;                       4482
011446  005266  000004            2$:     INC     4(SP)                           ; DIF                   4490
011452  020105                            CMP     R1,R5                           ; CUR.SUM,PRE.SUM       4492
011454  003404                            BLE     3$
011456  010105                            MOV     R1,R5                           ; CUR.SUM,PRE.SUM       4495
011460  016666  000006  000002            MOV     6(SP),2(SP)                     ; CUR.MOST,PRE.MOST     4496
011466  012701  000001            3$:     MOV     #1,R1                           ; *,CUR.SUM             4499
011472  005202                    4$:     INC     R2                              ; INDEX                 4475
011474  020204                    5$:     CMP     R2,R4                           ; INDEX,*
011476  003744                            BLE     1$
011500  005726                            TST     (SP)+                           ;                       4465
011502  020105                    6$:     CMP     R1,R5                           ; CUR.SUM,PRE.SUM       4511
011504  003403                            BLE     7$
011506  010105                            MOV     R1,R5                           ; CUR.SUM,PRE.SUM       4514
011510  016616  000004                    MOV     4(SP),(SP)                      ; CUR.MOST,PRE.MOST     4515
```

```
011514  012676  000026                7$:     MOV     (SP)+,@26(SP)                   ; PRE.MOST,MOST.OFTEN                   4518
011520  010576  000024                        MOV     R5,@24(SP)                      ; PRE.SUM,SUM.MOST                      4519
011524  012600                              MOV     (SP)+,R0                        ; DIF,*                                 4358
011526  005726                              TST     (SP)+                           ;                                       4357
011530  000207                              RTS     PC

; Routine Size:  65 words
; Maximum stack depth per invocation:  10 words
```

```
4522  routine S_ROWS (MOST_OFTEN, SUM_MOST, LST_REM_ENTRY) =
4523      begin
4524
4525  !++
4526  ! Functional Description:
4527  !       This global routine has the same functionality
4528  !       as search columns except that the copied
4529  !       remainder table is searched for row numbers.
4530  !
4531  ! Formal Parameters:
4532  !       MOST_OFTEN:
4533  !               Passed by reference and stores the most
4534  !               often found row address.
4535  !       SUM_OFTEN:
4536  !               Passed by reference and stores the sum
4537  !               of the most often found row address.
4538  !       LST_REM_ENTRY:
4539  !               Points to the last copied remainder entry
4540  !
4541  ! Implicit Inputs:
4542  !       MOST_OFTEN AND SUM_MOST which are passed by reference
4543  !       to this routine is returned to the caller by reference
4544  !       with the row number which appears most often and the
4545  !       number of times it appears.
4546  !
4547  ! Completion codes:
4548  !       The number of different row numbers which were found
4549  !       in the copied remainder table is returned to the caller.
4550  !
4551  ! Side Effects:
4552  !       none
4553  !--
4554
4555      local
4556          CUR_SUM,                    !Stores current sum of most often found row
4557          PRE_SUM,                    !Stores previous sum of mos. often found row
4558          CUR_MOST,                   !Stores current most often row found
4559          PRE_MOST,                   !Stores previous most often row found
4560          DIF;
4561
4562      !
4563      ! Start out the search by defining
4564      ! the current count, previous count
4565      ! and different count to be one.
4566      !
4567      ! Also define the first table entry
4568      ! to be the row number to be selected
4569      ! for blasting.
4570      !
4571      ! This is done in the event that there
4572      ! is only one table entry left to be
4573      ! selected for blasting.
```

```
4574      !
4575      CUR_SUM = 1;                                  !Current sum starts at one
4576      PRE_SUM = 1;                                  !Previous sum starts at one
4577      PRE_MOST = .COPIED_REM_TBL [ZERO, ROW];       !Previous most often starts at first table enrry
4578      DIF = 1;                                      !Different row count starts at one
4579
4580      !+
4581      ! If only one table entry is remaining
4582      ! in the table then return with the default
4583      ! values defined above as the routines return
4584      ! values.
4585      !-
4586
4587      if .LST_REM_ENTRY gtr ONE                     !First table entry is most often if table entry < 1
4588      then
4589          begin
4590          ROW_SORT (.LST_REM_ENTRY);                !Sort the row side of copied rem table
4591          !
4592          ! Index through the table and search for
4593          ! the row most often found and the number
4594          ! of time its found. Also search the table
4595          ! for the number of different row numbers
4596          ! found.
4597          !
4598
4599          incr index from 0 to .LST_REM_ENTRY - 2 do      !Find the sum, most often and different
4600              begin
4601              !
4602              ! Is this table position contents the same as the
4603              ! next table contents?
4604              !
4605
4606              if .COPIED_REM_TBL [.index, ROW] eql .COPIED_REM_TBL [.index + 1, ROW]
4607              then
4608                  begin
4609                  CUR_MOST = .COPIED_REM_TBL [.index, ROW];       !Most often is this entry
4610                  CUR_SUM = .CUR_SUM + 1;           !Increment the sum of most
4611                  end
4612              else
4613                  begin
4614                  DIF = .DIF + 1;                   !Increment the different row count
4615
4616                  if .CUR_SUM gtr .PRE_SUM          !Is the current sum > previous sum
4617                  then
4618                      begin
4619                      PRE_SUM = .CUR_SUM;           !Save the sum of the current most often
4620                      PRE_MOST = .CUR_MOST;         !Save the current most often row
4621                      end;
4622
4623                  CUR_SUM = 1;                      !Start current sum back to one
4624                  end;
4625
```

```
;        4626                end;
;        4627
;        4628            end;
;        4629
;        4630        !
;        4631        ! This test to see if the last set of
;        4632        ! unique row numbers in table is > the previous
;        4633        !
;        4634
;        4635        IF .cur_sum GTR .pre_sum                    !is the current sum > previous sum
;        4636        then
;        4637            begin
;        4638            PRE_SUM = .CUR_SUM;                     !Save the sum of the current most often
;        4639            PRE_MOST = .CUR_MOST;                   !Save the current most often row
;        4640            end;
;        4641
;        4642        .MOST_OFTEN = .PRE_MOST;                    !Return the most often found row
;        4643        .SUM_MOST = .PRE_SUM;                       !Return the sum of the most often found row
;        4644        return .DIF;                                !Return the number of different found rows
;        4645        end;


                                        .SBTTL  S.ROWS ROUTINE DECLARATIONS
011532  004167  000000G          S.ROWS: JSR     R1,$SAVE5                       ;                               4522
011536  024646                          CMP     -(SP),-(SP)
011540  012701  000001                  MOV     #1,R1                           ; *,CUR.SUM                     4575
011544  010105                          MOV     R1,R5                           ; *,PRE.SUM                     4576
011546  005046                          CLR     -(SP)                           ; PRE.MOST                      4577
011550  116716  001713'                MOVB    COPIED.REM.TBL+1,(SP)           ; *,PRE.MOST
011554  012766  000001  000002          MOV     #1,2(SP)                        ; *,DIF                         4578
011562  016604  000024                  MOV     24(SP),R4                       ; LST.REM.ENTRY,*               4587
011566  020427  000001                  CMP     R4,#1
011572  003444                          BLE     6$
011574  010446                          MOV     R4,-(SP)                        ;                               4590
011576  004767  177310                  JSR     PC,ROW.SORT
011602  162704  000002                  SUB     #2,R4                           ;                               4599
011606  005002                          CLR     R2                              ; INDEX
011610  000432                          BR      5$
011612  010203                  1$:     MOV     R2,R3                           ; INDEX,*                       4606
011614  006303                          ASL     R3
011616  010200                          MOV     R2,R0                           ; INDEX,*
011620  006300                          ASL     R0
011622  126360  001713'  001715'        CMPB    COPIED.REM.TBL+1(R3),COPIED.REM.TBL+3(R0) ;
011630  001007                          BNE     2$
011632  116366  001713'  000006        MOVB    COPIED.REM.TBL+1(R3),6(SP)      ; *,CUR.MOST                    4609
011640  105066  000007                  CLRB    7(SP)                           ; CUR.MOST
011644  005201                          INC     R1                              ; CUR.SUM                       4610
011646  000412                          BR      4$                              ;                               4606
011650  005266  000004          2$:     INC     4(SP)                           ; DIF                           4614
011654  020105                          CMP     R1,R5                           ; CUR.SUM,PRE.SUM               4616
011656  003404                          BLE     3$
011660  010105                          MOV     R1,R5                           ; CUR.SUM,PRE.SUM               4619
```

```
011662  016666  000006  000002                MOV     6(SP),2(SP)                     ; CUR.MOST,PRE.MOST                     4620
011670  012701  000001                3$:     MOV     #1,R1                           ; *,CUR.SUM                             4623
011674  005202                        4$:     INC     R2                              ; INDEX                                 4599
011676  020204                        5$:     CMP     R2,R4                           ; INDEX,*
011700  003744                                BLE     1$
011702  005726                                TST     (SP)+                           ;                                       4589
011704  020105                        6$:     CMP     R1,R5                           ; CUR.SUM,PRE.SUM                       4635
011706  003403                                BLE     7$
011710  010105                                MOV     R1,R5                           ; CUR.SUM,PRE.SUM                       4638
011712  016616  000004                        MOV     4(SP),(SP)                      ; CUR.MOST,PRE.MOST                     4639
011716  012676  000026                7$:     MOV     (SP)+,@26(SP)                   ; PRE.MOST,MOST.OFTEN                   4642
011722  010576  000024                        MOV     R5,@24(SP)                      ; PRE.SUM,SUM.MOST                      4643
011726  012600                                MOV     (SP)+,R0                        ; DIF,*                                 4523
011730  005726                                TST     (SP)+                           ;                                       4522
011732  000207                                RTS     PC

; Routine Size:  65 words
; Maximum stack depth per invocation:  10 words
```

```
;          4646  routine S_REM_TBL (BAD_CHIP, ALL_BAD) =
;          4647      begin
;          4648
;          4649  !++
;          4650  ! Functional Description:
;          4651  !       The remainder table is loaded with all the
;          4652  !       remaining single cell failures scattered
;          4653  !       through out this failing chip.
;          4654  !
;          4655  !       These scattered remaining row column pairs
;          4656  !       are transfered into the remainder table
;          4657  !       were they are interigated for the best possible
;          4658  !       blasting selection.
;          4659  !
;          4660  !       This global routine searches the remainder table of
;          4661  !       the remaining row column pairs and determines
;          4662  !       the best possible choice for blasting.
;          4663  !
;          4664  ! Formal Parameters:
;          4665  !       BAD_CHIP:
;          4666  !               Points the the failing chip presently
;          4667  !               being pm'ed.
;          4668  !       ALL_BAD:
;          4669  !               Counts how many all bad rows and columns
;          4670  !               have been found thus far.  If this count
;          4671  !               exceeds a count of 10 for any one chip
;          4672  !               then that chip is called an all bad chip and
;          4673  !               a 'condition A' message is printed.  If
;          4674  !               two all bad chips are discovered in the
;          4675  !               the same bank then a 'condition B' message
;          4676  !               is printed and further testing of this
;          4677  !               array is aborted.
;          4678  !
;          4679  ! Implicit Inputs:
;          4680  !       TMP_BLST_TBL
;          4681  !
;          4682  ! Implicit Outputs:
;          4683  !       The temporary blast table is loaded with a row
;          4684  !       or column address selected for blasting.
;          4685  !
;          4686  !       The number of all bad row columns found by this routine
;          4687  !       is added to the count in all_bad.
;          4688  !
;          4689  ! Completion codes:
;          4690  !       The value of all_bad which indicates how many all bad chips
;          4691  !       found thus far is returned.
;          4692  !
;          4693  ! Side Effects:
;          4694  !       none
;          4695  !--
;          4696
;          4697      local
```

```
4698            LST_REM_ENTRY,                  !Points to last entry into remainder table
4699            DIF_COLS,                       !Different no. of cols in remainder table
4700            DIF_ROWS,                       !Differnet no. of rows in remainder table
4701            ROW_MOST_OFTEN,                 !Most often found row in remainder table
4702            COL_MOST_OFTEN,                 !Most often found column in remainder table
4703            ROW_SUM,                        !Sum of the most often found row in remainder table
4704            COL_SUM;                        !Sum of the most often found column in remainder table
4705
4706        I_REM_TBL ();                       !Init the remainder and copied rem tables
4707        LST_REM_ENTRY = ZERO;               !Last rem entry starts at zero
4708        LST_REM_ENTRY = X_TO_REM_TBL (.LST_REM_ENTRY);
4709                                            !Xfer column count bable to remainder and copied tables
4710
4711        !+
4712        ! Repeat searching the remainder and copied
4713        ! remainder tables for the best choice of
4714        ! blasting until all remaining row/column
4715        ! pairs have been selected for blasting.
4716        !-
4717
4718        while .LST_REM_ENTRY gtr ZERO do    !Search the remainder and copied table until empty
4719            begin
4720            !
4721            ! If the all bad count reaches 10 then
4722            ! this chip is concidered to be all bad.
4723            ! End this routine and return the all bad
4724            ! count else continue searching the tables
4725            ! for the best choice for blasting.
4726            !
4727            ALL_BAD = .ALL_BAD + 1;         !Increment the all bad row / column count
4728
4729            if .ALL_BAD gtr 9               !Is the all bad count > 9
4730            then
4731                return .ALL_BAD             !Exit and return the count of all bad row/col
4732            else
4733                begin
4734                !
4735                ! Search the remainder and copied remainder tables and find:
4736                !
4737                !     1. Number of different columns found in each.
4738                !     2. Number of different rows found in each.
4739                !     3. The column number most often found.
4740                !     4. The number of times the most often column number was found.
4741                !     5. The row number most often found.
4742                !     6. The number of times the most row number was found.
4743                !
4744                DIF_COLS = S_COLUMNS (COL_MOST_OFTEN, COL_SUM, .LST_REM_ENTRY);
4745                                            !Find the columns different, most lften and sum
4746                DIF_ROWS = S_ROWS (ROW_MOST_OFTEN, ROW_SUM, .LST_REM_ENTRY);
4747                                            !Find the rows different, most foten and sum
4748                !
4749                ! If occurances of unigue row numbers occured
```

```
;          4750                ! more often then did columns numbers then choose
;          4751                ! rows for blasting else pick columns for blasting.
;          4752                !
;          4753
;          4754                if .DIF_ROWS leq .DIF_COLS          !Are different rows < different columns
;          4755                then
;          4756                    begin
;          4757                    TMP_BLST_TBL [.ALL_BAD, R_C] = SET_FLG; !Indicate this is a bad row
;          4758                    TMP_BLST_TBL [.ALL_BAD, R_C_NO] = .ROW_MOST_OFTEN;      !Load the failing row no.
;          4759                    TMP_BLST_TBL [.ALL_BAD, NIB_NO] = .BAD_CHIP74;  !Load the failing nibble  position
;          4760                    ROW_PURGE (.ROW_MOST_OFTEN, .ROW_SUM, .LST_REM_ENTRY);  !Purge this row from table
;          4761                    LST_REM_ENTRY = .LST_REM_ENTRY - .ROW_SUM;      !Realign the pointer
;          4762                    end
;          4763                else
;          4764                    begin
;          4765                    TMP_BLST_TBL [.ALL_BAD, R_C] = CLR_FLG; !Indicate this is a bad colums
;          4766                    TMP_BLST_TBL [.ALL_BAD, R_C_NO] = .COL_MOST_OFTEN;      !Load the failing column no.
;          4767                    TMP_BLST_TBL [.ALL_BAD, NIB_NO] = .BAD_CHIP74;  !Load the failing nibble position
;          4768    !
;          4769    ! VER CZMLCB ADDED '.' TO LST_REM_ENTRY
;          4770    !
;          4771                    COL_PURGE (.COL_MOST_OFTEN, .COL_SUM, .LST_REM_ENTRY);  !Purge column form talbe
;          4772                    LST_REM_ENTRY = .LST_REM_ENTRY - .COL_SUM;      !Realign pointer
;          4773                    end;
;          4774
;          4775                end;
;          4776
;          4777            end;
;          4778
;          4779        return .ALL_BAD;                        !Return the number of all bad rows/columns found
;          4780        end;


                                    .SBTTL  S.REM.TBL ROUTINE DECLARATIONS
011734  004167  000000G       S.REM.TBL:
                                    JSR     R1,$SAVE5                   ;                                   4646
011740  162706  000010              SUB     #10,SP
011744  004767  173230              JSR     PC,I.REM.TBL                ;                                   4706
011750  005002                      CLR     R2                          ; LST.REM.ENTRY                     4707
011752  005046                      CLR     -(SP)                       ;                                   4708
011754  004767  175644              JSR     PC,X.TO.REM.TBL
011760  010002                      MOV     R0,R2                       ; *,LST.REM.ENTRY
011762  005702          1$:         TST     R2                          ; LST.REM.ENTRY                     4718
011764  003543                      BLE     5$
011766  005266  000030              INC     30(SP)                      ; ALL.BAD                           4727
011772  016603  000030              MOV     30(SP),R3                   ; ALL.BAD,*                         4729
011776  020327  000011              CMP     R3,#11
012002  003403                      BLE     2$
012004  005726                      TST     (SP)+                       ;                                   4731
012006  010300                      MOV     R3,R0
012010  000534                      BR      6$
012012  012746  000012      2$:     MOV     #12,-(SP)                   ;                                   4744
```

```
012016  060616                 ADD     SP,(SP)                 ; COL.MOST.OFTEN,*
012020  012746  000012         MOV     #12,-(SP)
012024  060616                 ADD     SP,(SP)                 ; COL.SUM,*
012026  010246                 MOV     R2,-(SP)                ; LST.REM.ENTRY,*
012030  004767  177274         JSR     PC,S.COLUMNS
012034  010004                 MOV     R0,R4                   ; *,DIF.COLS
012036  012716  000012         MOV     #12,(SP)                ;                                       4746
012042  060616                 ADD     SP,(SP)                 ; ROW.MOST.OFTEN,*
012044  012746  000012         MOV     #12,-(SP)
012050  060616                 ADD     SP,(SP)                 ; ROW.SUM,*
012052  010246                 MOV     R2,-(SP)                ; LST.REM.ENTRY,*
012054  004767  177452         JSR     PC,S.ROWS
012060  010001                 MOV     R0,R1                   ; *,DIF.ROWS
012062  016616  000044         MOV     44(SP),(SP)             ; BAD.CHIP,*                            4759
012066  012746  000004         MOV     #4,-(SP)
012072  004767  000000G        JSR     PC,BL$DIV
012076  006303                 ASL     R3                      ;                                       4757
012100  062703  030346'        ADD     #TMP.BLST.TBL,R3
012104  020104                 CMP     R1,R4                   ; DIF.ROWS,DIF.COLS                     4754
012106  003034                 BGT     3$
012110  052713  100000         BIS     #100000,(R3)            ;                                       4757
012114  016605  000020         MOV     20(SP),R5               ; ROW.MOST.OFTEN,*                      4758
012120  006305                 ASL     R5
012122  006305                 ASL     R5
012124  006305                 ASL     R5
012126  006305                 ASL     R5
012130  042705  170017         BIC     #170017,R5
012134  042713  007760         BIC     #7760,(R3)
012140  050513                 BIS     R5,(R3)
012142  042700  177760         BIC     #177760,R0              ;                                       4759
012146  142713  000017         BICB    #17,(R3)
012152  150013                 BISB    R0,(R3)
012154  016646  000020         MOV     20(SP),-(SP)            ; ROW.MOST.OFTEN,*                      4760
012160  016646  000020         MOV     20(SP),-(SP)            ; ROW.SUM,*
012164  010246                 MOV     R2,-(SP)                ; LST.REM.ENTRY,*
012166  004767  176406         JSR     PC,ROW.PURGE
012172  166602  000024         SUB     24(SP),R2               ; ROW.SUM,LST.REM.ENTRY                 4761
012176  000433                 BR      4$                      ;                                       4754
012200  042713  100000 3$:     BIC     #100000,(R3)            ;                                       4765
012204  016605  000024         MOV     24(SP),R5               ; COL.MOST.OFTEN,*                      4766
012210  006305                 ASL     R5
012212  006305                 ASL     R5
012214  006305                 ASL     R5
012216  006305                 ASL     R5
012220  042705  170017         BIC     #170017,R5
012224  042713  007760         BIC     #7760,(R3)
012230  050513                 BIS     R5,(R3)
012232  042700  177760         BIC     #177760,R0              ;                                       4767
012236  142713  000017         BICB    #17,(R3)
012242  150013                 BISB    R0,(R3)
012244  016646  000024         MOV     24(SP),-(SP)            ; COL.MOST.OFTEN,*                      4771
012250  016646  000024         MOV     24(SP),-(SP)            ; COL.SUM,*
```

```
012254  010246                          MOV     R2,-(SP)                        ; LST.REM.ENTRY,*
012256  004767  176132                  JSR     PC,COL.PURGE
012262  166602  000030                  SUB     30(SP),R2                       ; COL.SUM,LST.REM.ENTRY                 4772
012266  062706  000022          4$:     ADD     #22,SP                          ;                                       4733
012272  000633                          BR      1$                              ;                                       4718
012274  005726                  5$:     TST     (SP)+                           ;                                       4646
012276  016600  000026                  MOV     26(SP),R0                       ; ALL.BAD,*                             4647
012302  062706  000010          6$:     ADD     #10,SP                          ;                                       4646
012306  000207                          RTS     PC

; Routine Size:  118 words
; Maximum stack depth per invocation:  20 words
```

```
4781  routine S_COL_CNT_TBL (BAD_CHIP, ALL_BAD) =
4782      begin
4783
4784  !++
4785  ! Functional Description:
4786  !       The column count table stores a count of
4787  !       adjacent failing columns when the error
4788  !       map is searched for bad rows.
4789  !
4790  !       This global routine searches the column count
4791  !       table for column counts greater than 10.
4792  !       If a count of greater than 10 is found then this is
4793  !       called an all bad column and the temp
4794  !       blast table is loaded with this columns
4795  !       number.  The column count table at this
4796  !       all bad column number is then cleared.
4797  !
4798  ! Formal Parameters:
4799  !       BAD_CHIP:
4800  !               Points a failing chip which is presently being
4801  !               pm'ed.
4802  !       ALL_BAD:
4803  !               Keeps count of how many all bad row and or columns
4804  !               are found for this chip.  If this count count
4805  !               exceeds 10 then this chip is called a all bad
4806  !               chip.
4807  !
4808  ! Implicit Inputs:
4809  !       TMP_BLST_TBL, COL_CNT_TBL
4810  !
4811  ! Implicit Outputs:
4812  !       The temporary blast table is loaded with a row
4813  !       or column address selected for blasting.
4814  !
4815  !       The number of all bad row columns found by this routine
4816  !       is added to the count in all_bad.
4817  !
4818  ! Completion codes:
4819  !       The value of all_bad which indicates how many all bad chips
4820  !       found thus far is returned.
4821  !
4822  ! Side Effects:
4823  !       none
4824  !--
4825
4826      !+
4827      ! Search through the column count table and
4828      ! find column numbers which were found bad
4829      ! more than 10 times.  Columns which errored
4830      ! more than 10 times will immediately by chosen
4831      ! for blasting.
4832      !-
```

```
;          4833
;          4834      incr COL_NUM from 0 to .MAX_CHIP_COL - 1 do         !Search thru the column table
;          4835          begin
;          4836          !
;          4837          ! See if this column number was found
;          4838          ! in error greater than 10 times.
;          4839          !
;          4840          ! If it was then up the all bad count,
;          4841          ! load the temporary blast table with this
;          4842          ! column number and clear this table position
;          4843          ! of this column number.
;          4844          !
;          4845
;          4846          if .COL_CNT_TBL [.COL_NUM] gtr 10       !Is this locations count > 10
;          4847          then
;          4848              begin
;          4849              ALL_BAD = .ALL_BAD + 1;             !Increment the all bad row/col count
;          4850              !
;          4851              ! If the all bad count is equal to
;          4852              ! 10 then exit this routine with the
;          4853              ! all bad count else load the temp blast
;          4854              ! with this column number.
;          4855              !
;          4856
;          4857              if .ALL_BAD gtr 9                   !Is the all bad count > 9
;          4858              then
;          4859                  return .ALL_BAD                 !Exit and return all bad count
;          4860              else
;          4861                  begin
;          4862                  TMP_BLST_TBL [.ALL_BAD, R_C] = CLR_FLG; !Indicate that this is a bad column
;          4863                  TMP_BLST_TBL [.ALL_BAD, R_C_NO] = .COL_NUM;     !Load the bad column number
;          4864                  TMP_BLST_TBL [.ALL_BAD, NIB_NO] = .BAD_CHIP/4;  !Load the failing nibble position
;          4865                  COL_CNT_TBL [.COL_NUM] = ZEROES;        !Clear the table of this bad column
;          4866                  end;
;          4867
;          4868              end;
;          4869
;          4870          end;
;          4871
;          4872      !
;          4873      ! If the all bad count did not reach
;          4874      ! 10 then do a normal exit here with
;          4875      ! the present value of all bad count.
;          4876      !
;          4877      return .ALL_BAD;                            !Exit and return the number of all bad rows/cols
;          4878      end;

                                        .SBTTL  S.COL.CNT.TBL ROUTINE DECLARATIONS
012310  004167  000000G          S.COL.CNT.TBL:
                                        JSR     R1,$SAVE2                       ;                       4781
012314  005002                          CLR     R2                              ; COL.NUM               4834
```

```
012316  000453                          BR      4$
012320  126227  001002' 000012  1$:     CMPB    COL.CNT.TBL(R2),#12             ; *(COL.NUM),*                  4846
012326  101446                          BLOS    3$
012330  005266  000010                  INC     10(SP)                          ; ALL.BAD                       4849
012334  016601  000010                  MOV     10(SP),R1                       ; ALL.BAD,*                     4857
012340  020127  000011                  CMP     R1,#11
012344  003402                          BLE     2$
012346  010100                          MOV     R1,R0                           ;                               4859
012350  000207                          RTS     PC
012352  006301                  2$:     ASL     R1                              ;                               4862
012354  062701  030346'                 ADD     #TMP.BLST.TBL,R1
012360  042711  100000                  BIC     #100000,(R1)
012364  010200                          MOV     R2,R0                           ; COL.NUM,*                     4863
012366  006300                          ASL     R0
012370  006300                          ASL     R0
012372  006300                          ASL     R0
012374  006300                          ASL     R0
012376  042700  170017                  BIC     #170017,R0
012402  042711  007760                  BIC     #7760,(R1)
012406  050011                          BIS     R0,(R1)
012410  016646  000012                  MOV     12(SP),-(SP)                    ; BAD.CHIP,*                    4864
012414  012746  000004                  MOV     #4,-(SP)
012420  004767  000000G                 JSR     PC,BL$DIV
012424  042700  177760                  BIC     #177760,R0
012430  142711  000017                  BICB    #17,(R1)
012434  150011                          BISB    R0,(R1)
012436  105062  001002'                 CLRB    COL.CNT.TBL(R2)                 ; *(COL.NUM)                    4865
012442  022626                          CMP     (SP)+,(SP)+                     ;                               4861
012444  005202                  3$:     INC     R2                              ; COL.NUM                       4834
012446  020267  040376'          4$:     CMP     R2,MAX.CHIP.COL                 ; COL.NUM,*
012452  002722                          BLT     1$
012454  016600  000010                  MOV     10(SP),R0                       ; ALL.BAD,*                     4782
012460  000207                          RTS     PC                              ;                               4781

; Routine Size:  53 words
; Maximum stack depth per invocation:  5 words
```

```
;        4879  routine S_ERROR_MAP (BAD_CHIP) =
;        4880      begin
;        4881
;        4882  !++
;        4883  ! Functional Description:
;        4884  !       The error map is loaded with all the
;        4885  !       failing row numbers at thier failing
;        4886  !       sector addresses.
;        4887  !
;        4888  !       This global routine searches the error map for
;        4889  !       occurances of bad rows.
;        4890  !
;        4891  !       When a bad row is found a bad row count
;        4892  !       is incremented and a bad column count
;        4893  !       at the adjacent failing column is also
;        4894  !       incremented.
;        4895  !
;        4896  !       If the row count exceeds 10 then this is
;        4897  !       called an all bad row and the row number is
;        4898  !       loaded into the temp blast table.  The adjacent
;        4899  !       failing column count is decremented by one.
;        4900  !
;        4901  !       If after searching the error map the bad row
;        4902  !       does not exceed 10 then the program goes on to
;        4903  !       the next row and the failing column count are
;        4904  !       left alone.
;        4905  !
;        4906  ! Formal Parameters:
;        4907  !       BAD_CHIP:
;        4908  !               Points to a failing chip presently being
;        4909  !               pm'ed.
;        4910  !
;        4911  ! Implicit Inputs:
;        4912  !       ERROR_MAP, COL_CNT_TBL, COL_PTR, TMP_BLST_TBL
;        4913  !
;        4914  ! Implicit Outputs:
;        4915  !       The temporary blast table is loaded with a row
;        4916  !       or column address selected for blasting.
;        4917  !
;        4918  !       The number of all bad row columns found by this routine
;        4919  !       is added to the count in all_bad.
;        4920  !
;        4921  ! Completion codes:
;        4922  !       The value of all_bad which indicates how many all bad chips
;        4923  !       found thus far is returned.
;        4924  !
;        4925  !       The error map is searched for failing rows addresses
;        4926  !       and they are cleared if set.
;        4927  !
;        4928  !       The column count table is incremented with the number
;        4929  !       of times column pairs are detected failing.
;        4930  !
```

```
4931  ! Side Effects:
4932  !       none
4933  !--
4934
4935      label
4936          A;                                  !Leave loop label
4937
4938      local
4939          WRD_INDEX,                          !Stores the error map word where this row resides
4940          BIT_INDEX,                          !Stores the bit in the error map word where this bit resides
4941          ROW_CNT,                            !Count of how many bad row found
4942          PTR,                                !Pointer into to the column pointer table
4943          COL_INDEX,                          !Stores the calculated adjacent failing column number
4944          QD_PAIR_LOOP,                       !Selects how many quadrants to test
4945          QD_NUM_SEARCH,                      !Selects how many quadrants to test
4946          QD_OFFSET,                          !Off set use in calculating which quadrant to test
4947          START_SEC,                          !Starting sector variable
4948          END_SEC,                            !Ending sector variable
4949          ALL_BAD;                            !Stores how many all bad chips are found
4950
4951      ALL_BAD = -1;                           !Start all bad at -1 this value gets returned
4952      I_COL_CNT_TBL ();                       !Init the column count table
4953      !
4954      ! Determine which type chip this
4955      ! is and define this routines run
4956      ! time parameters.
4957      !
4958
4959      if .BNK_NUM_SEC eql 127                 !Is this a 16k chip
4960      then
4961          begin
4962          QD_PAIR_LOOP = 0;                   !16k chips only have one quadrant to test
4963          QD_NUM_SEARCH = 0;
4964          end
4965      else
4966          begin
4967          QD_PAIR_LOOP = 1;                   !64k chips have four quadrants to test
4968          QD_NUM_SEARCH = 1;
4969          end;
4970
4971      QD_OFFSET = -2;                         !Start the offset off at -2
4972
4973      incr QD_LOOP from 0 to .QD_PAIR_LOOP do !Loop on two quadrant pairs
4974          begin
4975          QD_OFFSET = .QD_OFFSET + 2;         !Up the offset by two
4976
4977          incr ROW_NUM from 0 to 127 do       !Search all words in this sector
4978              begin
4979              WRD_INDEX = .ROW_NUM/16;        !Calculate which word this row is in
4980              BIT_INDEX = .ROW_NUM mod 16;    !Calculate which bit in word this row is in
4981  A :
4982              begin
```

```
;          4983
;          4984                incr QD_SEARCH from 0 + .QD_OFFSET to .QD_NUM_SEARCH + .QD_OFFSET do
;          4985                                                    !Search both quadrants in this pair
;          4986                    begin
;          4987
;          4988                    case .QD_SEARCH from 0 to 3 of  !Which quadrant are we in
;          4989                        set
;          4990
;          4991                        [0] :                       !We are in the first quadrant or just a 16k part
;          4992                            begin
;          4993                            ROW_CNT = ZERO;         !Row starts at 0 for the first quad search
;          4994                            PTR = -1;               !The pointer starts at -1
;          4995                            I_COL_PTR ();           !Init the column pointer table
;          4996                            START_SEC = 0;          !Quad 0 sectors start at 0
;          4997                            END_SEC = 127;          !Quad 0 end sector ends at 127
;          4998                            end;
;          4999
;          5000                        [1] :                       !We are in quadrant 1
;          5001                            begin
;          5002                            START_SEC = 256;        !Quad 1 sectors start at 256
;          5003                            END_SEC = 383;          !Quad 1 sectors end at 383
;          5004                            end;
;          5005
;          5006                        [2] :                       !We are in quadrant 2
;          5007                            begin
;          5008                            ROW_CNT = ZERO;         !Start the row count at zero
;          5009                            PTR = -1;               !Reset the pointer
;          5010                            I_COL_PTR ();           !Init the column pointer table
;          5011                            START_SEC = 128;        !Quad 2 sectors start at 128
;          5012                            END_SEC = 255;          !Quad 2 sectors end at 255
;          5013                            end;
;          5014
;          5015                        [3] :                       !We are in quad 3
;          5016                            begin
;          5017                            START_SEC = 384;        !Quad 3 sectors start at 384
;          5018                            END_SEC = 511;          !Quad 3 sectors end at 511
;          5019                            end;
;          5020                        tes;
;          5021
;          5022                    incr SEC_NUM from .START_SEC to .END_SEC do     !Search thru all sectors for this quad
;          5023                        begin
;          5024
;          5025                        if .ERROR_MAP [.SEC_NUM, .WRD_INDEX, .BIT_INDEX, ONE, ZERO] eql 1   !Is this row bit set
;          5026                        then
;          5027                            begin
;          5028                            ROW_CNT = .ROW_CNT + 1; !Up the bad row count
;          5029
;          5030                            if .ROW_CNT gtr 9       !Is there > then 10 bad rows
;          5031                            then
;          5032                                begin
;          5033                                ALL_BAD = .ALL_BAD + 1;     !Up the all bad count
;          5034
```

```
;       5035                    if .ALL_BAD gtr 9   !Is there > 10 all bad rows/columns
;       5036                    then
;       5037                        return .ALL_BAD !Exit and return the the count
;       5038                    else
;       5039                        begin
;       5040                        TMP_BLST_TBL [.ALL_BAD, R_C] = SET_FLG; !This is a bad row
;       5041
;       5042                        selectone .SEC_NUM of   !Find which row number this is
;       5043                            set
;       5044
;       5045                            [0 to 127, 256 to 383] :
;       5046                                TMP_BLST_TBL [.ALL_BAD, R_C_NO] = .ROW_NUM;
;       5047
;       5048                            [128 to 255, 384 to 511] :
;       5049                                TMP_BLST_TBL [.ALL_BAD, R_C_NO] = .ROW_NUM + 128;
;       5050                            tes;
;       5051
;       5052                        TMP_BLST_TBL [.ALL_BAD, NIB_NO] = .BAD_CHIP/4;  !Find which nibble its in
;       5053
;       5054                        incr index from 0 to .PTR do    !Decrement the column counts
;       5055                            COL_CNT_TBL [.COL_PTR [.index]] = .COL_CNT_TBL [.COL_PTR [.index]] - 1;
;       5056
;       5057                        if .QD_PAIR_LOOP eql 0  !Clear the quad of this roww
;       5058                        then
;       5059
;       5060                            incr CLR_SEC_ROW from 0 to 127 do
;       5061                                ERROR_MAP [.CLR_SEC_ROW, .WRD_INDEX, .BIT_INDEX, ONE, ZERO] = CLR_FLG
;       5062                        else
;       5063
;       5064                            if .QD_SEARCH leq 1
;       5065                            then
;       5066                                begin
;       5067
;       5068                                incr CLR_SEC_ROW from 0 to 127 do
;       5069                                    ERROR_MAP [.CLR_SEC_ROW, .WRD_INDEX, .BIT_INDEX, ONE, ZERO] =
;       5070                                    CLR_FLG;
;       5071
;       5072                                incr CLR_SEC_ROW from 256 to 383 do
;       5073                                    ERROR_MAP [.CLR_SEC_ROW, .WRD_INDEX, .BIT_INDEX, ONE, ZERO] =
;       5074                                    CLR_FLG;
;       5075
;       5076                                end
;       5077                            else
;       5078                                begin
;       5079
;       5080                                incr CLR_SEC_ROW from 128 to 255 do
;       5081                                    ERROR_MAP [.CLR_SEC_ROW, .WRD_INDEX, .BIT_INDEX, ONE, ZERO] =
;       5082                                    CLR_FLG;
;       5083
;       5084                                incr CLR_SEC_ROW from 384 to 511 do
;       5085                                    ERROR_MAP [.CLR_SEC_ROW, .WRD_INDEX, .BIT_INDEX, ONE, ZERO] =
;       5086                                    CLR_FLG;
```

```
;       5087
;       5088                                        end;
;       5089
;       5090                                    leave A;        !Leave the loop and do the next row number
;       5091                                    end;
;       5092
;       5093                                end
;       5094                            else
;       5095                                begin
;       5096
;       5097                                selectone .SEC_NUM of       !Increment the adjacent column count table
;       5098                                    set
;       5099
;       5100                                    [0 to 127] :
;       5101                                        begin
;       5102                                        COL_INDEX = .ROW_NUM + .SEC_NUM;
;       5103
;       5104                                        if .COL_INDEX geq 128 then COL_INDEX = .COL_INDEX - 128;
;       5105
;       5106                                        end;
;       5107
;       5108                                    [128 to 255] :
;       5109                                        begin
;       5110                                        COL_INDEX = .ROW_NUM + (.SEC_NUM - 128);
;       5111
;       5112                                        if .COL_INDEX geq 128 then COL_INDEX = .COL_INDEX - 128;
;       5113
;       5114                                        end;
;       5115
;       5116                                    [256 to 383] :
;       5117                                        begin
;       5118                                        COL_INDEX = .ROW_NUM + (.SEC_NUM - 128);
;       5119
;       5120                                        if .COL_INDEX geq 256 then COL_INDEX = .COL_INDEX - 128;
;       5121
;       5122                                        end;
;       5123
;       5124                                    [384 to 511] :
;       5125                                        begin
;       5126                                        COL_INDEX = .ROW_NUM + (.SEC_NUM - 256);
;       5127
;       5128                                        if .COL_INDEX geq 256 then COL_INDEX = .COL_INDEX - 128;
;       5129
;       5130                                        end;
;       5131                                    tes;
;       5132
;       5133                                COL_CNT_TBL [.COL_INDEX] = .COL_CNT_TBL [.COL_INDEX] + 1;
;       5134                                PTR = .PTR + 1;
;       5135                                COL_PTR [.PTR] = .COL_INDEX;
;       5136                                end;
;       5137
;       5138                            end;
```

```
;          5139
;          5140                        end;
;          5141
;          5142                    end;
;          5143
;          5144                end;
;          5145                end;
;          5146
;          5147            end;
;          5148
;          5149        return .ALL_BAD;
;          5150        end;


                                        .SBTTL  S.ERROR.MAP ROUTINE DECLARATIONS
012462  004167  000000G                S.ERROR.MAP:
                                        JSR     R1,$SAVE5                       ;                                       4879
012466  162706  000032                  SUB     #32,SP
012472  012766  177777  000010          MOV     #-1,10(SP)                      ; *,ALL.BAD                             4951
012500  004767  172454                  JSR     PC,I.COL.CNT.TBL                ;                                       4952
012504  026727  040416' 000177          CMP     BNK.NUM.SEC,#177                ;                                       4959
012512  001005                          BNE     1$
012514  005066  000022                  CLR     22(SP)                          ; QD.PAIR.LOOP                          4962
012520  005066  000030                  CLR     30(SP)                          ; QD.NUM.SEARCH                         4963
012524  000406                          BR      2$                              ;                                       4959
012526  012766  000001  000022  1$:     MOV     #1,22(SP)                       ; *,QD.PAIR.LOOP                        4967
012534  012766  000001  000030          MOV     #1,30(SP)                       ; *,QD.NUM.SEARCH                       4968
012542  012766  177776  000020  2$:     MOV     #-2,20(SP)                      ; *,QD.OFFSET                           4971
012550  005066  000026                  CLR     26(SP)                          ; QD.LOOP                               4973
012554  000167  001502                  JMP     43$
012560  062766  000002  000020  3$:     ADD     #2,20(SP)                       ; *,QD.OFFSET                           4975
012566  016666  000030  000016          MOV     30(SP),16(SP)                   ; QD.NUM.SEARCH,*                       4984
012574  066666  000020  000016          ADD     20(SP),16(SP)                   ; QD.OFFSET,*
012602  005003                          CLR     R3                              ; ROW.NUM                               4977
012604  010346                  4$:     MOV     R3,-(SP)                        ; ROW.NUM,*                             4979
012606  012746  000020                  MOV     #20,-(SP)
012612  004767  000000G                 JSR     PC,BL$DIV
012616  010066  000012                  MOV     R0,12(SP)                       ; *,WRD.INDEX
012622  010316                          MOV     R3,(SP)                         ; ROW.NUM,*                             4980
012624  012746  000020                  MOV     #20,-(SP)
012630  004767  000000G                 JSR     PC,BL$MOD
012634  010066  000012                  MOV     R0,12(SP)                       ; *,BIT.INDEX
012640  016666  000026  000006          MOV     26(SP),6(SP)                    ; QD.OFFSET,QD.SEARCH                   4984
012646  005366  000006                  DEC     6(SP)                           ; QD.SEARCH
012652  000167  001340                  JMP     40$
012656  016605  000006          5$:     MOV     6(SP),R5                        ; QD.SEARCH,*                           4988
012662  006305                          ASL     R5
012664  066507  012670'                 ADD     6$(R5),PC
012670  000010                  6$:     .WORD   7$-6$
012672  000042                          .WORD   8$-6$
012674  000060                          .WORD   9$-6$
012676  000114                          .WORD   10$-6$
```

```
012700  005066  000032                  7$:     CLR     32(SP)                  ; ROW.CNT                       4993
012704  012766  177777  000010                  MOV     #-1,10(SP)              ; *,PTR                         4994
012712  004767  172316                          JSR     PC,I.COL.PTR            ;                               4995
012716  005066  000022                          CLR     22(SP)                  ; START.SEC                     4996
012722  012766  000177  000020                  MOV     #177,20(SP)             ; *,END.SEC                     4997
012730  000433                                  BR      11$                     ;                               4988
012732  012766  000400  000022          8$:     MOV     #400,22(SP)             ; *,START.SEC                   5002
012740  012766  000577  000020                  MOV     #577,20(SP)             ; *,END.SEC                     5003
012746  000424                                  BR      11$                     ;                               4988
012750  005066  000032                  9$:     CLR     32(SP)                  ; ROW.CNT                       5008
012754  012766  177777  000010                  MOV     #-1,10(SP)              ; *,PTR                         5009
012762  004767  172246                          JSR     PC,I.COL.PTR            ;                               5010
012766  012766  000200  000022                  MOV     #200,22(SP)             ; *,START.SEC                   5011
012774  012766  000377  000020                  MOV     #377,20(SP)             ; *,END.SEC                     5012
013002  000406                                  BR      11$                     ;                               4988
013004  012766  000600  000022          10$:    MOV     #600,22(SP)             ; *,START.SEC                   5017
013012  012766  000777  000020                  MOV     #777,20(SP)             ; *,END.SEC                     5018
013020  016605  000022                  11$:    MOV     22(SP),R5               ; START.SEC,SEC.NUM             5022
013024  005305                                  DEC     R5                      ; SEC.NUM
013026  000167  001150                  12$:    JMP     39$
013032  010500                          13$:    MOV     R5,R0                   ; SEC.NUM,*                     5025
013034  006300                                  ASL     R0
013036  006300                                  ASL     R0
013040  006300                                  ASL     R0
013042  066600  000014                          ADD     14(SP),R0               ; WRD.INDEX,*
013046  006300                                  ASL     R0
013050  062700  010230'                        ADD     #ERROR.MAP,R0
013054  010046                                  MOV     R0,-(SP)
013056  016646  000014                          MOV     14(SP),-(SP)            ; BIT.INDEX,*
013062  012746  000001                          MOV     #1,-(SP)
013066  005046                                  CLR     -(SP)
013070  004767  000000G                        JSR     PC,BL$GT2
013074  062706  000010                          ADD     #10,SP
013100  005300                                  DEC     R0
013102  001351                                  BNE     12$
013104  005266  000032                          INC     32(SP)                  ; ROW.CNT                       5028
013110  026627  000032  000011                  CMP     32(SP),#11              ; ROW.CNT,*                     5030
013116  003002                                  BGT     14$
013120  000167  000654                          JMP     33$
013124  005266  000016                  14$:    INC     16(SP)                  ; ALL.BAD                       5033
013130  026627  000016  000011                  CMP     16(SP),#11              ; ALL.BAD,*                     5035
013136  003404                                  BLE     15$
013140  062706  000006                          ADD     #6,SP                   ;                               5037
013144  000167  001126                          JMP     44$
013150  016604  000016                  15$:    MOV     16(SP),R4               ; ALL.BAD,*                     5040
013154  006304                                  ASL     R4
013156  062704  030346'                        ADD     #TMP.BLST.TBL,R4
013162  052714  100000                          BIS     #100000,(R4)
013166  005705                                  TST     R5                      ; SEC.NUM                       5042
013170  002403                                  BLT     16$
013172  020527  000177                          CMP     R5,#177                 ; SEC.NUM,*
013176  003406                                  BLE     17$
```

```
013200  020527  000400          16$:    CMP     R5,#400                         ; SEC.NUM,*
013204  002405                          BLT     18$
013206  020527  000577                  CMP     R5,#577                         ; SEC.NUM,*
013212  003002                          BGT     18$
013214  010302                  17$:    MOV     R3,R2                           ; ROW.NUM,*                 5046
013216  000417                          BR      21$
013220  020527  000200          18$:    CMP     R5,#200                         ; SEC.NUM,*                 5042
013224  002403                          BLT     19$
013226  020527  000377                  CMP     R5,#377                         ; SEC.NUM,*
013232  003406                          BLE     20$
013234  020527  000600          19$:    CMP     R5,#600                         ; SEC.NUM,*
013240  002417                          BLT     22$
013242  020527  000777                  CMP     R5,#777                         ; SEC.NUM,*
013246  003014                          BGT     22$
013250  010302                  20$:    MOV     R3,R2                           ; ROW.NUM,*                 5049
013252  062702  000200                  ADD     #200,R2
013256  006302                  21$:    ASL     R2
013260  006302                          ASL     R2
013262  006302                          ASL     R2
013264  006302                          ASL     R2
013266  042702  170017                  BIC     #170017,R2
013272  042714  007760                  BIC     #7760,(R4)
013276  050214                          BIS     R2,(R4)
013300  016646  000056          22$:    MOV     56(SP),-(SP)                    ; BAD.CHIP,*                5052
013304  012746  000004                  MOV     #4,-(SP)
013310  004767  000000G                 JSR     PC,BL$DIV
013314  042700  177760                  BIC     #177760,R0
013320  142714  000017                  BICB    #17,(R4)
013324  150014                          BISB    R0,(R4)
013326  005002                          CLR     R2                              ; INDEX                     5054
013330  000412                          BR      24$
013332  005004                  23$:    CLR     R4                              ;                           5055
013334  156204  002222'                BISB    COL.PTR(R2),R4                  ; *(INDEX),*
013340  005000                          CLR     R0
013342  156400  001002'                BISB    COL.CNT.TBL(R4),R0
013346  005300                          DEC     R0
013350  110064  001002'                MOVB    R0,COL.CNT.TBL(R4)
013354  005202                          INC     R2                              ; INDEX                     5054
013356  020266  000014          24$:    CMP     R2,14(SP)                       ; INDEX,PTR
013362  003763                          BLE     23$
013364  005766  000034                  TST     34(SP)                          ; QD.PAIR.LOOP              5057
013370  001031                          BNE     26$
013372  005004                          CLR     R4                              ; CLR.SEC.ROW               5060
013374  010400                  25$:    MOV     R4,R0                           ; CLR.SEC.ROW,*             5061
013376  006300                          ASL     R0
013400  006300                          ASL     R0
013402  006300                          ASL     R0
013404  066600  000020                  ADD     20(SP),R0                       ; WRD.INDEX,*
013410  006300                          ASL     R0
013412  062700  010230'                ADD     #ERROR.MAP,R0
013416  010046                          MOV     R0,-(SP)
013420  016646  000020                  MOV     20(SP),-(SP)                    ; BIT.INDEX,*
```

```
013424  012746  000001                          MOV     #1,-(SP)
013430  005046                                  CLR     -(SP)
013432  004767  000000G                         JSR     PC,BL$PU2
013436  062706  000010                          ADD     #10,SP
013442  005204                                  INC     R4                      ; CLR.SEC.ROW                   5060
013444  020427  000177                          CMP     R4,#177                 ; CLR.SEC.ROW,*
013450  003751                                  BLE     25$
013452  000550                                  BR      32$                     ;                               5057
013454  026627  000012  000001        26$:      CMP     12(SP),#1               ; QD.SEARCH,*                   5064
013462  003062                                  BGT     29$
013464  005004                                  CLR     R4                      ; CLR.SEC.ROW                   5068
013466  010400                          27$:    MOV     R4,R0                   ; CLR.SEC.ROW,*                 5069
013470  006300                                  ASL     R0
013472  006300                                  ASL     R0
013474  006300                                  ASL     R0
013476  066600  000020                          ADD     20(SP),R0               ; WRD.INDEX,*
013502  006300                                  ASL     R0
013504  062700  010230'                         ADD     #ERROR.MAP,R0
013510  010046                                  MOV     R0,-(SP)
013512  016646  000020                          MOV     20(SP),-(SP)            ; BIT.INDEX,*
013516  012746  000001                          MOV     #1,-(SP)
013522  005046                                  CLR     -(SP)
013524  004767  000000G                         JSR     PC,BL$PU2
013530  062706  000010                          ADD     #10,SP
013534  005204                                  INC     R4                      ; CLR.SEC.ROW                   5068
013536  020427  000177                          CMP     R4,#177                 ; CLR.SEC.ROW,*
013542  003751                                  BLE     27$
013544  012704  000400                          MOV     #400,R4                 ; *,CLR.SEC.ROW                 5072
013550  010400                          28$:    MOV     R4,R0                   ; CLR.SEC.ROW,*                 5073
013552  006300                                  ASL     R0
013554  006300                                  ASL     R0
013556  006300                                  ASL     R0
013560  066600  000020                          ADD     20(SP),R0               ; WRD.INDEX,*
013564  006300                                  ASL     R0
013566  062700  010230'                         ADD     #ERROR.MAP,R0
013572  010046                                  MOV     R0,-(SP)
013574  016646  000020                          MOV     20(SP),-(SP)            ; BIT.INDEX,*
013600  012746  000001                          MOV     #1,-(SP)
013604  005046                                  CLR     -(SP)
013606  004767  000000G                         JSR     PC,BL$PU2
013612  062706  000010                          ADD     #10,SP
013616  005204                                  INC     R4                      ; CLR.SEC.ROW                   5072
013620  020427  000577                          CMP     R4,#577                 ; CLR.SEC.ROW,*
013624  003751                                  BLE     28$
013626  000462                                  BR      32$                     ;                               5064
013630  012704  000200                  29$:    MOV     #200,R4                 ; *,CLR.SEC.ROW                 5080
013634  010400                          30$:    MOV     R4,R0                   ; CLR.SEC.ROW,*                 5081
013636  006300                                  ASL     R0
013640  006300                                  ASL     R0
013642  006300                                  ASL     R0
013644  066600  000020                          ADD     20(SP),R0               ; WRD.INDEX,*
013650  006300                                  ASL     R0
```

```
013652  062700  010230'                ADD     #ERROR.MAP,R0
013656  010046                         MOV     R0,-(SP)
013660  016646  000020                 MOV     20(SP),-(SP)            ; BIT.INDEX,*
013664  012746  000001                 MOV     #1,-(SP)
013670  005046                         CLR     -(SP)
013672  004767  000000G                JSR     PC,BL$PU2
013676  062706  000010                 ADD     #10,SP
013702  005204                         INC     R4                      ; CLR.SEC.ROW                   5080
013704  020427  000377                 CMP     R4,#377                 ; CLR.SEC.ROW,*
013710  003751                         BLE     30$
013712  012704  000600                 MOV     #600,R4                 ; *,CLR.SEC.ROW                 5084
013716  010400              31$:       MOV     R4,R0                   ; CLR.SEC.ROW,*                 5085
013720  006300                         ASL     R0
013722  006300                         ASL     R0
013724  006300                         ASL     R0
013726  066600  000020                 ADD     20(SP),R0               ; WRD.INDEX,*
013732  006300                         ASL     R0
013734  062700  010230'                ADD     #ERROR.MAP,R0
013740  010046                         MOV     R0,-(SP)
013742  016646  000020                 MOV     20(SP),-(SP)            ; BIT.INDEX,*
013746  012746  000001                 MOV     #1,-(SP)
013752  005046                         CLR     -(SP)
013754  004767  000000G                JSR     PC,BL$PU2
013760  062706  000010                 ADD     #10,SP
013764  005204                         INC     R4                      ; CLR.SEC.ROW                   5084
013766  020427  000777                 CMP     R4,#777                 ; CLR.SEC.ROW,*
013772  003751                         BLE     31$
013774  022626              32$:       CMP     (SP)+,(SP)+             ;                               5090
013776  000517                         BR      41$
014000  005705              33$:       TST     R5                      ; SEC.NUM                       5097
014002  002411                         BLT     34$
014004  020527  000177                 CMP     R5,#177                 ; SEC.NUM,*
014010  003006                         BGT     34$
014012  010501                         MOV     R5,R1                   ; SEC.NUM,COL.INDEX             5102
014014  060301                         ADD     R3,R1                   ; ROW.NUM,COL.INDEX
014016  020127  000200                 CMP     R1,#200                 ; COL.INDEX,*                   5104
014022  002457                         BLT     38$
014024  000454                         BR      37$
014026  020527  000200              34$:       CMP     R5,#200         ; SEC.NUM,*                     5097
014032  002414                         BLT     35$
014034  020527  000377                 CMP     R5,#377                 ; SEC.NUM,*
014040  003011                         BGT     35$
014042  010504                         MOV     R5,R4                   ; SEC.NUM,*                     5110
014044  060304                         ADD     R3,R4                   ; ROW.NUM,*
014046  010401                         MOV     R4,R1                   ; *,COL.INDEX
014050  162701  000200                 SUB     #200,R1                 ; *,COL.INDEX
014054  020127  000200                 CMP     R1,#200                 ; COL.INDEX,*                   5112
014060  002440                         BLT     38$
014062  000435                         BR      37$
014064  020527  000400              35$:       CMP     R5,#400         ; SEC.NUM,*                     5097
014070  002414                         BLT     36$
014072  020527  000577                 CMP     R5,#577                 ; SEC.NUM,*
```

```
014076  003011                          BGT     36$
014100  010504                          MOV     R5,R4                   ; SEC.NUM,*                     5118
014102  060304                          ADD     R3,R4                   ; ROW.NUM,*
014104  010401                          MOV     R4,R1                   ; *,COL.INDEX
014106  162701  000200                  SUB     #200,R1                 ; *,COL.INDEX
014112  020127  000400                  CMP     R1,#400                 ; COL.INDEX,*                   5120
014116  002421                          BLT     38$
014120  000416                          BR      37$
014122  020527  000600          36$:    CMP     R5,#600                 ; SEC.NUM,*                     5097
014126  002415                          BLT     38$
014130  020527  000777                  CMP     R5,#777                 ; SEC.NUM,*
014134  003012                          BGT     38$
014136  010504                          MOV     R5,R4                   ; SEC.NUM,*                     5126
014140  060304                          ADD     R3,R4                   ; ROW.NUM,*
014142  010401                          MOV     R4,R1                   ; *,COL.INDEX
014144  162701  000400                  SUB     #400,R1                 ; *,COL.INDEX
014150  020127  000400                  CMP     R1,#400                 ; COL.INDEX,*                   5128
014154  002402                          BLT     38$
014156  162701  000200          37$:    SUB     #200,R1                 ; *,COL.INDEX
014162  105261  001002'        38$:    INCB    COL.CNT.TBL(R1)         ; *(COL.INDEX)                  5133
014166  005266  000010                  INC     10(SP)                  ; PTR                           5134
014172  016604  000010                  MOV     10(SP),R4               ; PTR,*                         5135
014176  110164  002222'                MOVB    R1,COL.PTR(R4)          ; COL.INDEX,*
014202  005205                  39$:    INC     R5                      ; SEC.NUM                       5022
014204  020566  000020                  CMP     R5,20(SP)               ; SEC.NUM,END.SEC
014210  003002                          BGT     40$
014212  000167  176614                  JMP     13$
014216  005266  000006          40$:    INC     6(SP)                   ; QD.SEARCH                     4984
014222  026666  000006  000024          CMP     6(SP),24(SP)            ; QD.SEARCH,*
014230  003002                          BGT     41$
014232  000167  176420                  JMP     5$
014236  062706  000006          41$:    ADD     #6,SP                   ;                               4978
014242  005203                          INC     R3                      ; ROW.NUM                       4977
014244  020327  000177                  CMP     R3,#177                 ; ROW.NUM,*
014250  003002                          BGT     42$
014252  000167  176326                  JMP     4$
014256  005266  000026          42$:    INC     26(SP)                  ; QD.LOOP                       4973
014262  026666  000026  000022  43$:    CMP     26(SP),22(SP)           ; QD.LOOP,QD.PAIR.LOOP
014270  003002                          BGT     44$
014272  000167  176262                  JMP     3$
014276  016600  000010          44$:    MOV     10(SP),R0               ; ALL.BAD,*                     4880
014302  062706  000032                  ADD     #32,SP                  ;                               4879
014306  000207                          RTS     PC

; Routine Size:  459 words
; Maximum stack depth per invocation:  28 words
```

```
;          5151  routine S_CHIP_TBL (START) =
;          5152      begin
;          5153
;          5154  !++
;          5155  ! Functional Description:
;          5156  !       This global routine searches the bad chip table
;          5157  !       looking for failing chips.  When a failing
;          5158  !       chip is found the failing chip number is
;          5159  !       returned.  If no failing chip is found
;          5160  !       then a -1 is returned.
;          5161  !
;          5162  ! Formal Parameters:
;          5163  !       START:
;          5164  !               Indicates where to start searching
;          5165  !               in the bad chip table
;          5166  !
;          5167  ! Implicit Inputs:
;          5168  !       CHIP_TBL
;          5169  !
;          5170  ! Implicit Outputs:
;          5171  !       none
;          5172  !
;          5173  ! Completion codes:
;          5174  !       The number of the next failing chip to be PM'ed
;          5175  !       is returned or a -1 is returned if no failing chips
;          5176  !       remain in this bank to be tested.
;          5177  !
;          5178  ! Side Effects:
;          5179  !       none
;          5180  !--
;          5181
;          5182      !+
;          5183      ! Search the chip table and find the
;          5184      ! next failing chip to PM.  If one
;          5185      ! is found then return to the caller
;          5186      ! that chip number else return a -1
;          5187      ! to indicate that all the failing
;          5188      ! chips have been PM'ed
;          5189      !-
;          5190
;          5191      incr TBL_INDEX from .START to 38 do         !Search thru the chip table
;          5192
;          5193          if .CHIP_TBL [.TBL_INDEX, FAULT] then return .TBL_INDEX;        !Exit and return ndex
;          5194
;          5195      return -1;                                  !Exit and return -1 if no chip faults are detected
;          5196      end;


                                        .SBTTL  S.CHIP.TBL ROUTINE DECLARATIONS
014310  010146                  S.CHIP.TBL:
                                        MOV     R1,-(SP)                        ;                                       5151
014312  016600  000004                  MOV     4(SP),R0                        ; START.TBL.INDEX                       5191
```

```
014316  005300                          DEC     R0                      ; TBL.INDEX
014320  000405                          BR      2$
014322  010001                  1$:     MOV     R0,R1                   ; TBL.INDEX,*                   5193
014324  006301                          ASL     R1
014326  005761  030230'                  TST     CHIP.TBL(R1)
014332  100406                          BMI     3$
014334  005200                  2$:     INC     R0                      ; TBL.INDEX                     5191
014336  020027  000046                  CMP     R0,#46                  ; TBL.INDEX,*
014342  003767                          BLE     1$
014344  012700  177777                  MOV     #-1,R0                  ;                               5152
014350  012601                  3$:     MOV     (SP)+,R1                ;                               5151
014352  000207                          RTS     PC

; Routine Size:  18 words
; Maximum stack depth per invocation:  2 words
```

```
;  5197  routine RD_OLD_PROM_DATA (ARR$BNK_SEL) : novalue =
;  5198      begin
;  5199
;  5200  !++
;  5201  ! Functional Description:
;  5202  !       In order to perform Prom Maintenance on an
;  5203  !       array module the old prom data stored in the
;  5204  !       tested array must be read and saved into the
;  5205  !       error map where the new prom data found by
;  5206  !       this program can be OR'ed with it.
;  5207  !
;  5208  !       This global routine will read the old prom data
;  5209  !       stored in prom on the tested array into
;  5210  !       the error map.
;  5211  !
;  5212  ! Formal Parameters:
;  5213  !       ARR$BNK_SEL:
;  5214  !               Stores the array and bank select address.
;  5215  !
;  5216  ! Implicit Inputs:
;  5217  !       ERROR_MAP, RET_STATUS
;  5218  !
;  5219  ! Implicit Outputs:
;  5220  !       The error map is remapped to look exactly like
;  5221  !       the blast table and the old prom data in this
;  5222  !       tested array is read into it.
;  5223  !
;  5224  ! Completion codes:
;  5225  !       none
;  5226  !
;  5227  ! Side Effects:
;  5228  !       none
;  5229  !--
;  5230
;  5231      local
;  5232          PROM_ADRS,                          !Stores the built prom address
;  5233          FINISH;                             !Incr loop ending variable
;  5234
;  5235      !+
;  5236      ! To conserve memory space the ML-11 exercisers
;  5237      ! write and read buffers are redefined to be a
;  5238      ! contigious blockvector of 4 blocks of 512 words
;  5239      ! each.  This map declaration maps the exercisers
;  5240      ! I/O buffers into this structure.
;  5241      !-
;  5242
;  5243      map
;  5244          ERROR_MAP : blockvector [4, 512] volatile;      !Map the error_map the same as the blast table
;  5245
;  5246      !
;  5247      ! Define how many rows and column to read
;  5248      ! out of the on board UV proms.
```

```
5249        !
5250
5251        if .BNK_NUM_SEC eql 127 then FINISH = 127 else FINISH = 255;        !Select the loops ending
5252
5253        RD_PROM_MODE;                                                       !Enable prom reads
5254        PROM_ADRS = ZEROES;                                                 !Clear the prom address
5255        PROM_ADRS<11, 4> = .ARR$BNK_SEL<.ARR_SEL_POS, ARR$SEL_SIZE>;        !Select the array
5256
5257        !+
5258        ! Read all old prom data stored in the on board
5259        ! proms and store the prom data in the
5260        ! error map at its respective error map
5261        ! location.
5262        !-
5263
5264        incr SEL_BNK from 0 to 3 do                                         !Read prom data for all four banks on this array
5265            begin
5266            PROM_ADRS<10, 1> = ZERO;                                        !The first loop reads row prom data
5267            PROM_ADRS<8, 2> = .SEL_BNK;                                     !Select the proms bank position
5268
5269            !+
5270            ! Within this bank of the on board proms
5271            ! read all row prom data and store it into the
5272            ! error map at its respective error map
5273            ! location.
5274            !-
5275
5276            incr ROW_NUM from 0 to .FINISH do                               !Read out all the prom row data
5277                begin
5278                PROM_ADRS<0, 8> = .ROW_NUM;                                 !Select the row address
5279                WRT_RH (MLPA, PA_REG, .PROM_ADRS);                          !Write the address to the mlpa reg
5280                DELAY (ONE_US);                                             !Wait for prom data to clock into mlpd
5281                ERROR_MAP [.SEL_BNK, .ROW_NUM, FULL_WRD] = .ML_ADDR [MLPD, PD_REG];
5282                                                                            !Put the prom data into the error map
5283                end;
5284
5285        PROM_ADRS<10, 1> = ONE;                                             !This loop will read out the prom column data
5286
5287        !+
5288        ! Within this bank of the on board proms
5289        ! read all column prom data and store it into the
5290        ! error map at its respective error map
5291        ! location.
5292        !-
5293
5294        incr COL_NUM from 0 to .FINISH do                                   !Read out all the prom column data
5295            begin
5296            PROM_ADRS<0, 8> = .COL_NUM;                                     !Select the column address
5297            WRT_RH (MLPA, PA_REG, .PROM_ADRS);                              !Write the address to the mlpa reg
5298            DELAY (ONE_US);                                                 !Wait for prom data to clock into mlpd
5299            ERROR_MAP [.SEL_BNK, .COL_NUM + .COL_BASE, FULL_WRD] = .ML_ADDR [MLPD, PD_REG];
5300                                                                            !Load the error map with the prom data
```

```
;         5301                end;
;         5302
;         5303            end;
;         5304
;         5305        CLR_MBUS;                       !Clear out the prom read mode
;         5306        end;


                                        .GLOBL  L$DLY


                                        .SBTTL  RD.OLD.PROM.DAT ROUTINE DECLARATIONS
014354  004167  000000G          RD.OLD.PROM.DAT:
                                        JSR     R1,$SAVE5                       ;                       5197
014360  162706  000022                  SUB     #22,SP
014364  026727  040416' 000177          CMP     BNK.NUM.SEC,#177                ;                       5251
014372  001003                          BNE     1$
014374  012716  000177                  MOV     #177,(SP)                       ; *,FINISH
014400  000402                          BR      2$
014402  012716  000377          1$:     MOV     #377,(SP)                       ; *,FINISH
014406  016701  040374'         2$:     MOV     ML.ADDR,R1
014412  016166  000024  000016          MOV     24(R1),16(SP)                   ; *,ML.REG
014420  012700  000040                  MOV     #40,R0                          ; *,MLREG
014424  016701  040374'                 MOV     ML.ADDR,R1
014430  010061  000024                  MOV     R0,24(R1)                       ; MLREG,*
014434  005002                          CLR     R2                              ; PROM.ADRS             5254
014436  012746  000042                  MOV     #42,-(SP)                       ;                       5255
014442  060616                          ADD     SP,(SP)                         ; ARR$BNK.SEL,*
014444  016746  040412'                 MOV     ARR.SEL.POS,-(SP)
014450  012746  000004                  MOV     #4,-(SP)
014454  005046                          CLR     -(SP)
014456  004767  000000G                 JSR     PC,BL$GT2
014462  000300                          SWAB    R0
014464  006300                          ASL     R0
014466  006300                          ASL     R0
014470  006300                          ASL     R0
014472  042700  103777                  BIC     #103777,R0
014476  042702  074000                  BIC     #74000,R2                       ; *,PROM.ADRS
014502  050002                          BIS     R0,R2                           ; *,PROM.ADRS
014504  005005                          CLR     R5                              ; SEL.BNK               5264
014506  010504                  3$:     MOV     R5,R4                           ; SEL.BNK,*             5267
014510  000304                          SWAB    R4
014512  042704  176377                  BIC     #176377,R4
014516  042702  003400                  BIC     #3400,R2                        ; *,PROM.ADRS
014522  050402                          BIS     R4,R2                           ; *,PROM.ADRS
014524  010500                          MOV     R5,R0                           ; SEL.BNK,*             5281
014526  000300                          SWAB    R0
014530  105000                          CLRB    R0
014532  006300                          ASL     R0
014534  005004                          CLR     R4                              ; ROW.NUM               5276
014536  000444                          BR      9$
014540  105002                  4$:     CLRB    R2                              ; PROM.ADRS             5278
```

```
014542  150402                          BISB    R4,R2                   ; ROW.NUM,PROM.ADRS
014544  016701  040374'                 MOV     ML.ADDR,R1              ;                                       5279
014550  016166  000020  000024          MOV     20(R1),24(SP)           ; *,ML.REG
014556  010203                          MOV     R2,R3                   ; PROM.ADRS,MLREG
014560  016701  040374'                 MOV     ML.ADDR,R1
014564  010361  000020                  MOV     R3,20(R1)               ; MLREG,*
014570  012703  000001                  MOV     #1,R3                   ; *,$$TMP2                              5280
014574  001411                  5$:     BEQ     8$
014576  016701  000000G                 MOV     L$DLY,R1                ; *,$$TMP1
014602  001404                          BEQ     7$
014604  005066  000030          6$:     CLR     30(SP)                  ; $$TMP
014610  005301                          DEC     R1                      ; $$TMP1
014612  001374                          BNE     6$
014614  005303                  7$:     DEC     R3                      ; $$TMP2
014616  000766                          BR      5$
014620  010003                  8$:     MOV     R0,R3                   ;                                       5281
014622  060403                          ADD     R4,R3                   ; ROW.NUM,*
014624  006303                          ASL     R3
014626  016701  040374'                 MOV     ML.ADDR,R1
014632  016166  000046  000022          MOV     46(R1),22(SP)           ; *,ML.REG
014640  016663  000022  010230'         MOV     22(SP),ERROR.MAP(R3)    ; ML.REG,*
014646  005204                          INC     R4                      ; ROW.NUM                               5276
014650  020466  000010          9$:     CMP     R4,10(SP)               ; ROW.NUM,FINISH
014654  003731                          BLE     4$
014656  052702  002000                  BIS     #2000,R2                ; *,PROM.ADRS                           5285
014662  005003                          CLR     R3                      ; COL.NUM                               5294
014664  000446                          BR      15$
014666  105002                  10$:    CLRB    R2                      ; PROM.ADRS                             5296
014670  150302                          BISB    R3,R2                   ; COL.NUM,PROM.ADRS
014672  016701  040374'                 MOV     ML.ADDR,R1              ;                                       5297
014676  016166  000020  000020          MOV     20(R1),20(SP)           ; *,ML.REG
014704  010204                          MOV     R2,R4                   ; PROM.ADRS,MLREG
014706  016701  040374'                 MOV     ML.ADDR,R1
014712  010461  000020                  MOV     R4,20(R1)               ; MLREG,*
014716  012704  000001                  MOV     #1,R4                   ; *,$$TMP2                              5298
014722  001411                  11$:    BEQ     14$
014724  016701  000000G                 MOV     L$DLY,R1                ; *,$$TMP1
014730  001404                          BEQ     13$
014732  005066  000030          12$:    CLR     30(SP)                  ; $$TMP
014736  005301                          DEC     R1                      ; $$TMP1
014740  001374                          BNE     12$
014742  005304                  13$:    DEC     R4                      ; $$TMP2
014744  000766                          BR      11$
014746  010304                  14$:    MOV     R3,R4                   ; COL.NUM,*                             5299
014750  066704  040372'                 ADD     COL.BASE,R4
014754  060004                          ADD     R0,R4
014756  006304                          ASL     R4
014760  016701  040374'                 MOV     ML.ADDR,R1
014764  016166  000046  000016          MOV     46(R1),16(SP)           ; *,ML.REG
014772  016664  000016  010230'         MOV     16(SP),ERROR.MAP(R4)    ; ML.REG,*
015000  005203                          INC     R3                      ; COL.NUM                               5294
015002  020366  000010          15$:    CMP     R3,10(SP)               ; COL.NUM,FINISH
```

```
015006  003727                          BLE     10$
015010  005205                          INC     R5                      ; SEL.BNK                       5264
015012  020527  000003                  CMP     R5,#3                   ; SEL.BNK,*
015016  003633                          BLE     3$
015020  016701  040374'                 MOV     ML.ADDR,R1              ;                               5303
015024  016166  000010  000014          MOV     10(R1),14(SP)           ; *,ML.REG
015032  016600  000014                  MOV     14(SP),R0               ; ML.REG,MLREG
015036  152700  000040                  BISB    #40,R0                  ; *,MLREG
015042  016701  040374'                 MOV     ML.ADDR,R1
015046  010061  000010                  MOV     R0,10(R1)               ; MLREG,*
015052  016701  040374'                 MOV     ML.ADDR,R1
015056  016166  000010  000012          MOV     10(R1),12(SP)           ; *,ML.REG
015064  016600  000012                  MOV     12(SP),R0               ; ML.REG,MLREG
015070  016705  040400'                 MOV     ML.DUT,R5
015074  042705  177770                  BIC     #177770,R5
015100  142700  000007                  BICB    #7,R0                   ; *,MLREG
015104  050500                          BIS     R5,R0                   ; *,MLREG
015106  016701  040374'                 MOV     ML.ADDR,R1
015112  010061  000010                  MOV     R0,10(R1)               ; MLREG,*
015116  062706  000032                  ADD     #32,SP                  ;                               5197
015122  000207                          RTS     PC

; Routine Size:  180 words
; Maximum stack depth per invocation:  19 words
```

```
;          5307  routine OR_OLD_NEW_PD (BNK_NUM) : novalue =
;          5308      begin
;          5309
;          5310  !++
;          5311  ! Functional Description:
;          5312  !       Once the nibble offsets have been counted and
;          5313  !       offsets > 14 do not exist then the
;          5314  !       old prom data stored in the error map
;          5315  !       at this bank is OR'ed into the blast
;          5316  !       table where the check sums can then be calculated.
;          5317  !
;          5318  ! Formal Parameters:
;          5319  !       ARR$BNK_SEL:
;          5320  !               Stores array and bank select address.
;          5321  !
;          5322  ! Implicit Inputs:
;          5323  !       BLAST_TBL, ERROR_MAP
;          5324  !
;          5325  ! Implicit Outputs:
;          5326  !       The old prom data in the error map is OR'ed
;          5327  !       into their respective blast table locations.
;          5328  !
;          5329  ! Completion codes:
;          5330  !       none
;          5331  !
;          5332  ! Side Effects:
;          5333  !       none
;          5334  !--
;          5335
;          5336      !+
;          5337      ! To conserve memory space the ML-11 exercisers
;          5338      ! write and read buffers are redefined to be a
;          5339      ! contigious blockvector of 4 blocks of 512 words
;          5340      ! each.  This map declaration maps the exercisers
;          5341      ! I/O buffers into this structure.
;          5342      !-
;          5343
;          5344      map
;          5345          ERROR_MAP : blockvector [4, 512] volatile;      !Map the error map like the blast tasble
;          5346
;          5347      !+
;          5348      ! Logically 'OR' all non zero blast table locations
;          5349      ! with respective error map locations.
;          5350      !-
;          5351
;          5352      incr CNT from 0 to .MAX_CHIP_COL*2 - 1 do   !Or all non zero blast table locations
;          5353          begin
;          5354          !
;          5355          ! Logical 'OR' this blast table location with
;          5356          ! its error map counter part if it is non zero.
;          5357          !
;          5358
```

```
;          5359          if .BLAST_TBL [.BNK_NUM, .CNT, FULL_WRD] neq ZERO       !Is this location not zero
;          5360          then
;          5361              BLAST_TBL [.BNK_NUM, .CNT, FULL_WRD] = .BLAST_TBL [.BNK_NUM, .CNT, FULL_WRD] or .ERROR_MAP [
;          5362                  .BNK_NUM, .CNT, FULL_WRD];      !Or this location with adjacent error map location
;          5363
;          5364          end;
;          5365
;          5366      end;


                                        .SBTTL  OR.OLD.NEW.PD ROUTINE DECLARATIONS
015124  004167  000000G          OR.OLD.NEW.PD:
                                        JSR     R1,$SAVE4                       ;                               5307
015130  016704  040376'                 MOV     MAX.CHIP.COL,R4                 ;                               5352
015134  006304                          ASL     R4
015136  016600  000014                  MOV     14(SP),R0                       ; BNK.NUM,*                     5359
015142  000300                          SWAB    R0
015144  105000                          CLRB    R0
015146  006300                          ASL     R0
015150  005001                          CLR     R1                              ; CNT                           5352
015152  000414                          BR      3$
015154  010003                  1$:     MOV     R0,R3                           ;                               5359
015156  060103                          ADD     R1,R3                           ; CNT,*
015160  006303                          ASL     R3
015162  012702  030372'                 MOV     #BLAST.TBL,R2
015166  060302                          ADD     R3,R2
015170  005712                          TST     (R2)
015172  001403                          BEQ     2$
015174  056363  010230' 030372'        BIS     ERROR.MAP(R3),BLAST.TBL(R3)     ;                               5361
015202  005201                  2$:     INC     R1                              ; CNT                           5352
015204  020104                  3$:     CMP     R1,R4                           ; CNT,*
015206  002762                          BLT     1$
015210  000207                          RTS     PC                              ;                               5307

; Routine Size:  27 words
; Maximum stack depth per invocation:  5 words
```

```
;        5367  routine BIT_CLR_OLD_NEW (BNK_NUM) : novalue =
;        5368      begin
;        5369
;        5370  !++
;        5371  ! Functional Description:
;        5372  !       When a failing chip is searched for newly
;        5373  !       failing rows and columns there is always
;        5374  !       the possibility that old bad rows or columns
;        5375  !       may also fail.  In fact we hope they do
;        5376  !       so to give us a performancd history of the chip.
;        5377  !
;        5378  !       These failing rows and columns if any
;        5379  !       must be clear out from the blast table
;        5380  !       to allow the counting of bad nibble off-
;        5381  !       sets.
;        5382  !
;        5383  !       This global routine bit clears the error map with
;        5384  !       the blast table at the selected bank.
;        5385  !
;        5386  ! Formal Parameters:
;        5387  !       BNK_NUM:
;        5388  !               Selected bank number for bit clearing.
;        5389  !
;        5390  ! Implicit Inputs:
;        5391  !       ERROR_MAP, BLAST_TBL
;        5392  !
;        5393  ! Implicit Outputs:
;        5394  !       The blast table is cleared of old prom data
;        5395  !       occurances.
;        5396  !
;        5397  ! Completion codes:
;        5398  !       none
;        5399  !
;        5400  ! Side Effects:
;        5401  !       none
;        5402  !--
;        5403
;        5404      !+
;        5405      ! To conserve memory space the ML-11 exercisers
;        5406      ! write and read buffers are redefined to be a
;        5407      ! contigious blockvector of 4 blocks of 512 words
;        5408      ! each.  This map declaration maps the exercisers
;        5409      ! I/O buffers into this structure.
;        5410      !-
;        5411
;        5412      map
;        5413          ERROR_MAP : blockvector [4, 512] volatile;      !Map the error map like the blast table
;        5414
;        5415      !+
;        5416      ! Bit clear all blast table locations with their
;        5417      ! error map counter parts.
;        5418      !-
```

```
;          5419
;          5420      incr CNT from 0 to .MAX_CHIP_COL*2 - 1 do   !Bit clear blast table with error map at this bank
;          5421          begin
;          5422          BLAST_TBL [.BNK_NUM, .CNT, FULL_WRD] = ( not .ERROR_MAP [.BNK_NUM, .CNT, FULL_WRD]) and .BLAST_TBL [
;          5423              .BNK_NUM, .CNT, FULL_WRD];
;          5424          end;
;          5425
;          5426      end;


                                        .SBTTL  BIT.CLR.OLD.NEW ROUTINE DECLARATIONS
015212  004167  000000G          BIT.CLR.OLD.NEW:
                                        JSR     R1,$SAVE3                       ;                                       5367
015216  016703  040376'                MOV     MAX.CHIP.COL,R3                 ;                                       5420
015222  006303                        ASL     R3
015224  016600  000012                MOV     12(SP),R0                       ; BNK.NUM,*                             5422
015230  000300                        SWAB    R0
015232  105000                        CLRB    R0
015234  006300                        ASL     R0
015236  005001                        CLR     R1                              ; CNT                                   5420
015240  000407                        BR      2$
015242  010002                  1$:   MOV     R0,R2                           ;                                       5422
015244  060102                        ADD     R1,R2                           ; CNT,*
015246  006302                        ASL     R2
015250  046262  010230' 030372'       BIC     ERROR.MAP(R2),BLAST.TBL(R2)
015256  005201                        INC     R1                              ; CNT                                   5420
015260  020103                  2$:   CMP     R1,R3                           ; CNT,*
015262  002767                        BLT     1$
015264  000207                        RTS     PC                              ;                                       5367

; Routine Size:  22 words
; Maximum stack depth per invocation:  4 words
```

```
;       5427  routine IN_BLAST_TBL (ARR$BNK_SEL, TSTED_BNK) : novalue =
;       5428      begin
;       5429
;       5430  !++
;       5431  ! Functional Description:
;       5432  !       This global routine first clears the blast table
;       5433  !       of any old bad row or column data that might
;       5434  !       have been found during the searching for new
;       5435  !       failing rows and columns.
;       5436  !
;       5437  !       The selected bank is then searched to see if
;       5438  !       any additional failing rows or columns were
;       5439  !       found.
;       5440  !
;       5441  !       If no additional failing row columns were
;       5442  !       found then the bank is not further tested.
;       5443  !
;       5444  !       If additional failing row or columns were
;       5445  !       found then the blast table and the error
;       5446  !       map is search for nibble offset > 14.
;       5447  !
;       5448  !
;       5449  ! Formal Parameters:
;       5450  !       ARR$BNK_SEL:
;       5451  !               Stores array and bank select address
;       5452  !
;       5453  !       TSTED_BNK:
;       5454  !               Indicates how many banks per array module
;       5455  !               is to be pm'ed.
;       5456  !
;       5457  ! Implicit Inputs:
;       5458  !       ERROR_MAP, BLAST_TBL, BAD_BNK_REG, DEGRADE_MOD_REG
;       5459  !
;       5460  ! Implicit Outputs:
;       5461  !       1. If nibble offsets exceed 14 then bad row or
;       5462  !          column address in the blast table are cleared
;       5463  !          until nibble offsets equal 14.
;       5464  !
;       5465  !       2. A banks bad bank register bit is set if the
;       5466  !          bank has additional error to mask.
;       5467  !
;       5468  !       3. A banks degrage mode register bit is set if
;       5469  !          nibble offsets greater than 14 are detected.
;       5470  !
;       5471  ! Completion codes:
;       5472  !       none
;       5473  !
;       5474  ! Side Effects:
;       5475  !       none
;       5476  !--
;       5477  !
;       5478      local
```

```
5479            QD_0_SUM,                       !Quadrant 0 offset sum = sectors 0-127
5480            QD_1_SUM,                       !Quadrant 1 offset sum = sectors 128-255
5481            QD_2_SUM,                       !Quadrant 2 offset sum = sectors 256-383
5482            QD_3_SUM,                       !Quadrant 3 offset sum = sectors 384-511
5483            ROW_0_127_OFF,                  !Offsets count for rows 0-127
5484            ROW_128_255_OFF,                !Offset count for rows 128-255
5485            COL_0_127_OFF,                  !Offset count for columns 0-127
5486            COL_128_255_OFF,                !Offset count for column 128-255
5487            BNK_NUM,                        !Stores bank selector
5488            ARR_NUM,                        !Stores array selector
5489            OVER_FLOW;                      !Stores offset counts which are > 14
5490
5491        !+
5492        ! To conserve memory space the ML-11 exercisers
5493        ! write and read buffers are redefined to be a
5494        ! contigious blockvector of 4 blocks of 512 words
5495        ! each.  This map declaration maps the exercisers
5496        ! I/O buffers into this structure.
5497        !-
5498
5499        map
5500            ERROR_MAP : blockvector [4, 512] volatile;      !Map the error map like the blast table
5501
5502        I_ERROR_MAP ();                             !Init the error map before we store the prom data in there
5503        RD_OLD_PROM_DATA (.ARR$BNK_SEL);            !Read the old prom data into the error map
5504        BNK_NUM = .ARR$BNK_SEL<.BNK_SEL_POS, BNK$SEL_SIZE>; !Select the bank to interigate
5505        ARR_NUM = .ARR$BNK_SEL<.ARR_SEL_POS, ARR$SEL_SIZE>; !Select the array of this bank
5506
5507        incr SEL_BNK from 0 to .TSTED_BNK - 1 do    !Interigate the pm'ed banks
5508            begin
5509            BIT_CLR_OLD_NEW (.BNK_NUM);             !Clear the old failing rows/cols from the blast table
5510
5511            if S_BLAST_TBL (.BNK_NUM)               !Are there any new errors in this bank
5512            then
5513                begin
5514                BAD_BNK_REG [.BNK_NUM] = SET_FLG;   !Flag that this is a bad bank
5515
5516                incr NIB_NUM from 0 to 9 do         !Search thru all nibbles in this bank
5517                    begin
5518                    ROW_0_127_OFF = ZERO;           !Init the row count
5519                    ROW_128_255_OFF = ZERO;         !Init the row count
5520                    COL_0_127_OFF = ZERO;           !Init the column count
5521                    COL_128_255_OFF = ZERO;         !Init the column count
5522
5523                    incr ROW_NUM from 0 to 127 do   !Sum the number of failing rows 0-127
5524                        begin
5525
5526                        if .BLAST_TBL [.BNK_NUM, .ROW_NUM, .NIB_NUM, 1, 0]
5527                        then
5528                            ROW_0_127_OFF = .ROW_0_127_OFF + 1;
5529
5530                        if .ERROR_MAP [.BNK_NUM, .ROW_NUM, .NIB_NUM, 1, 0]
```

```
5531                        then
5532                            ROW_0_127_OFF = .ROW_0_127_OFF + 1;
5533
5534                        end;
5535
5536                    incr COL_NUM from 0 to 127 do   !Sum the number of failing columns 0-127
5537                        begin
5538
5539                        if .BLAST_TBL [.BNK_NUM, .COL_NUM + .COL_BASE, .NIB_NUM, 1, 0]
5540                        then
5541                            COL_0_127_OFF = .COL_0_127_OFF + 1;
5542
5543                        if .ERROR_MAP [.BNK_NUM, .COL_NUM + .COL_BASE, .NIB_NUM, 1, 0]
5544                        then
5545                            COL_0_127_OFF = .COL_0_127_OFF + 1;
5546
5547                        end;
5548
5549                    if .BNK_NUM_SEC gtr 127         !Is this a 64k part
5550                    then
5551                        begin
5552
5553                        incr ROW_NUM from 128 to 255 do     !Sum number of failing rows 128-255
5554                            begin
5555
5556    !
5557    ! Ver czmlcb added '.' to ROW_NUM
5558    !
5559                            if .BLAST_TBL [.BNK_NUM, .ROW_NUM, .NIB_NUM, 1, 0]
5560                            then
5561                                ROW_128_255_OFF = .ROW_128_255_OFF + 1;
5562
5563                            if .ERROR_MAP [.BNK_NUM, .ROW_NUM, .NIB_NUM, 1, 0]
5564                            then
5565                                ROW_128_255_OFF = .ROW_128_255_OFF + 1;
5566
5567                            end;
5568
5569                        incr COL_NUM from 128 to 255 do     !Sum number of failing col 128-255
5570                            begin
5571
5572                            if .BLAST_TBL [.BNK_NUM, .COL_NUM + .COL_BASE, .NIB_NUM, 1, 0]
5573                            then
5574                                COL_128_255_OFF = .COL_128_255_OFF + 1;
5575
5576                            if .ERROR_MAP [.BNK_NUM, .COL_NUM + .COL_BASE, .NIB_NUM, 1, 0]
5577                            then
5578                                COL_128_255_OFF = .COL_128_255_OFF + 1;
5579
5580                            end;
5581
5582                        QD_1_SUM = .ROW_128_255_OFF + .COL_0_127_OFF;       !Sum number of failures in quad 1
```

```
;          5583                    QD_2_SUM = .ROW_0_127_OFF + .COL_128_255_OFF;       !Sum number of failures inquad 2
;          5584                    QD_3_SUM = .ROW_128_255_OFF + .COL_128_255_OFF;     !Sum number of failures in quad 3
;          5585                    end;
;          5586
;          5587                QD_0_SUM = .ROW_0_127_OFF + .COL_0_127_OFF;     !Sum number of failures in quad 0
;          5588
;          5589                if .QD_0_SUM gtr 14             !Is quad 0 offsets > 14
;          5590                then
;          5591                    begin
;          5592                    ERRSF (ERR_7, CON_C_MSG, 0);        !Report the error
;          5593                    PRINTB (A_B_N_PRINT, .ARR_NUM, .BNK_NUM, .NIB_NUM); !Print where
;          5594                    DEGRADE_MOD_REG [.BNK_NUM] = SET_FLG;       !Indicate this is a degraded bank
;          5595                    OVER_FLOW = .QD_0_SUM - 14; !Calculate the difference
;          5596
;          5597                    incr ROW_NUM from 0 to 127 do       !Delete row offsets until eql 14
;          5598
;          5599                        if .BLAST_TBL [.BNK_NUM, .ROW_NUM, .NIB_NUM, 1, 0]      !Is this row bad
;          5600                        then
;          5601                            begin
;          5602                            BLAST_TBL [.BNK_NUM, .ROW_NUM, .NIB_NUM, 1, 0] = CLR_FLG;   !Clear this bad row
;          5603                            OVER_FLOW = .OVER_FLOW - 1; !Decrement the difference
;          5604                            QD_2_SUM = .QD_2_SUM - 1;   !Delete other sums using this row
;          5605
;          5606                            if .OVER_FLOW eql ZERO then exitloop;       !Exit if difference eql 0
;          5607
;          5608                            end;
;          5609
;          5610                    if .OVER_FLOW neq ZERO      !Did deleting bad row get the difference to 14
;          5611                    tnen
;          5612                        begin
;          5613
;          5614                        incr COL_NUM from 0 to 127 do   !Delete bad column until offsets are at 14
;          5615
;          5616                            if .BLAST_TBL [.BNK_NUM, .COL_NUM + .COL_BASE, .NIB_NUM, 1, 0]
;          5617                            then
;          5618                                begin
;          5619                                BLAST_TBL [.BNK_NUM, .COL_NUM + .COL_BASE, .NIB_NUM, 1, 0] = CLR_FLG;
;          5620                                OVER_FLOW = .OVER_FLOW - 1;     !Decrement the difference
;          5621                                QD_1_SUM = .QD_1_SUM - 1;       !Delete this column from other sums
;          5622
;          5623                                if .OVER_FLOW eql ZERO then exitloop;   !Exit if difference is at 0
;          5624
;          5625                                end;
;          5626
;          5627                        end;
;          5628
;          5629                    end;
;          5630
;          5631                if .BNK_NUM_SEC gtr 127         !Is this a 64k part
;          5632                then
;          5633                    begin
;          5634
```

```
5635                    if .QD_1_SUM gtr 14          !Is quad 1 sum > 14
5636                    then
5637                        begin
5638                        ERRSF (ERR_7, CON_C_MSG, 0);    !Report the error
5639                        PRINTB (A_B_N_PRINT, .ARR_NUM, .BNK_NUM, .NIB_NUM);     !Tell where
5640                        DEGRADE_MOD_REG [.BNK_NUM] = SET_FLG;   !Indicate this is a degraded bank
5641                        OVER_FLOW = .QD_1_SUM - 14;     !Calculate the difference
5642
5643                        incr ROW_NUM from 128 to 255 do         !Delete bad rows until offsets are at 14
5644
5645                            if .BLAST_TBL [.BNK_NUM, .ROW_NUM, .NIB_NUM, 1, 0]  !Is this a bad row
5646                            then
5647                                begin
5648                                BLAST_TBL [.BNK_NUM, .ROW_NUM, .NIB_NUM, 1, 0] = CLR_FLG;
5649                                OVER_FLOW = .OVER_FLOW - 1;     !Decrement the difference
5650                                QD_3_SUM = .QD_3_SUM - 1;       !Delete other sums of this bad row
5651
5652                                if .OVER_FLOW eql ZERO then exitloop;   !Exit when difference is at 0
5653
5654                                end;
5655
5656                        if .OVER_FLOW neq ZERO  !Is this quad offset at 14 now
5657                        then
5658                            begin
5659
5660                            incr COL_NUM from 0 to 127 do       !Delete bad columns until offsets are at 14
5661
5662                                if .BLAST_TBL [.BNK_NUM, .COL_NUM + .COL_BASE, .NIB_NUM, 1, 0]
5663                                then
5664                                    begin
5665                                    BLAST_TBL [.BNK_NUM, .COL_NUM + .COL_BASE, .NIB_NUM, 1, 0] = CLR_FLG;
5666                                    OVER_FLOW = .OVER_FLOW - 1; !Decrement difference
5667
5668                                    if .OVER_FLOW eql ZERO then exitloop;       !Exit when at 0
5669
5670                                    end;
5671
5672                            end;
5673
5674                        end;
5675
5676                    if .QD_2_SUM gtr 14          !Is this quad offsets > 14
5677                    then
5678                        begin
5679                        ERRSF (ERR_7, CON_C_MSG, 0);    !Report the error
5680                        PRINTB (A_B_N_PRINT, .ARR_NUM, .BNK_NUM, .NIB_NUM);     !Report where
5681                        DEGRADE_MOD_REG [.BNK_NUM] = SET_FLG;   !Indicate this is a degraded bank
5682                        OVER_FLOW = .QD_2_SUM - 14;     !Calculate the difference
5683
5684                        incr ROW_NUM from 0 to 127 do   !Delete bad rows until offset are at 14
5685
5686                            if .BLAST_TBL [.BNK_NUM, .ROW_NUM, .NIB_NUM, 1, 0]  !Is this a bad row
```

```
5687                                    then
5688                                        begin
5689                                        BLAST_TBL [.BNK_NUM, .ROW_NUM, .NIB_NUM, 1, 0] = CLR_FLG;
5690                                        OVER_FLOW = .OVER_FLOW - 1;     !Decrement the difference
5691
5692                                        if .OVER_FLOW eql ZERO then exitloop;   !Exit when at 0
5693
5694                                        end;
5695
5696                                    if .OVER_FLOW neq ZERO  !Is quad offsets at 14 yet
5697                                    then
5698                                        begin
5699
5700                                        incr COL_NUM from 128 to 255 do     !Delete bad columns until offsets at 14
5701
5702                                            if .BLAST_TBL [.BNK_NUM, .COL_NUM + .COL_BASE, .NIB_NUM, 1, 0]
5703                                            then
5704                                                begin
5705                                                BLAST_TBL [.BNK_NUM, .COL_NUM + .COL_BASE, .NIB_NUM, 1, 0] = CLR_FLG;
5706                                                QD_3_SUM = .QD_3_SUM - 1;   !Delete other sums of this column
5707                                                OVER_FLOW = .OVER_FLOW - 1; !Decrement the differnece
5708
5709                                                if .OVER_FLOW eql ZERO then exitloop;       !Exit when at 0
5710
5711                                                end;
5712
5713                                        end;
5714
5715                                    end;
5716
5717                                if .QD_3_SUM gtr 14         !Is this quad offsets > 14
5718                                then
5719                                    begin
5720                                    ERRSF (ERR_7, CON_C_MSG, 0);    !Report the error
5721                                    PRINTB (A_B_N_PRINT, .ARR_NUM, .BNK_NUM, .NIB_NUM);     !Print where
5722                                    DEGRADE_MOD_REG [.BNK_NUM] = SET_FLG;   !Indicate this is a degraded bank
5723                                    OVER_FLOW = .QD_3_SUM - 14;     !Calculate the difference
5724
5725                                    incr ROW_NUM from 128 to 255 do         !Delete bad rows until offset are at 14
5726
5727                                        if .BLAST_TBL [.BNK_NUM, .ROW_NUM, .NIB_NUM, 1, 0]  !Is this a bad row
5728                                        then
5729                                            begin
5730                                            BLAST_TBL [.BNK_NUM, .ROW_NUM, .NIB_NUM, 1, 0] = CLR_FLG;       !Clear
5731                                            OVER_FLOW = .OVER_FLOW - 1;     !Decrement the difference
5732
5733                                            if .OVER_FLOW eql ZERO then exitloop;   !Exit when at 0
5734
5735                                            end;
5736
5737                                    if .OVER_FLOW neq ZERO  !Is quad offsets at 14 yet
5738                                    then
```

```
;       5739                            begin
;       5740
;       5741                            incr COL_NUM from 128 to 255 do     !Delete bad column until offset at 14
;       5742
;       5743                                if .BLAST_TBL [.BNK_NUM, .COL_NUM + .COL_BASE, .NIB_NUM, 1, 0]
;       5744                                then
;       5745                                    begin
;       5746                                    BLAST_TBL [.BNK_NUM, .COL_NUM + .COL_BASE, .NIB_NUM, 1, 0] = CLR_FLG;
;       5747                                    OVER_FLOW = .OVER_FLOW - 1; !Decrement difference
;       5748
;       5749                                    if .OVER_FLOW eql ZERO then exitloop;       !Exit when at 0
;       5750
;       5751                                    end;
;       5752
;       5753                            end;
;       5754
;       5755                        end;
;       5756
;       5757                    end;
;       5758
;       5759                end;
;       5760
;       5761            OR_OLD_NEW_PD (.BNK_NUM);           !Or the old prom data with new prom data
;       5762            end;
;       5763
;       5764        BNK_NUM = .BNK_NUM + 1;                 !Increment to the next bank
;       5765        end;
;       5766
;       5767    end;


                        .SBTTL  IN.BLAST.TBL ROUTINE DECLARATIONS
015266  004167  000000G IN.BLAST.TBL:
                        JSR     R1,$SAVE5                   ;                                   5427
015272  162706  000030          SUB     #30,SP
015276  004767  167752          JSR     PC,I.ERROR.MAP      ;                                   5502
015302  016646  000050          MOV     50(SP),-(SP)        ; ARR$BNK.SEL,*                     5503
015306  004767  177042          JSR     PC,RD.OLD.PROM.DAT
015312  012716  000052          MOV     #52,(SP)            ;                                   5504
015316  060616                  ADD     SP,(SP)             ; ARR$BNK.SEL,*
015320  016746  040414'         MOV     BNK.SEL.POS,-(SP)
015324  012746  000002          MOV     #2,-(SP)
015330  005046                  CLR     -(SP)
015332  004767  000000G         JSR     PC,BL$GT2
015336  010066  000010          MOV     R0,10(SP)           ; *,BNK.NUM
015342  012716  000060          MOV     #60,(SP)            ;                                   5505
015346  060616                  ADD     SP,(SP)             ; ARR$BNK.SEL,*
015350  016746  040412'         MOV     ARR.SEL.POS,-(SP)
015354  012746  000004          MOV     #4,-(SP)
015360  005046                  CLR     -(SP)
015362  004767  000000G         JSR     PC,BL$GT2
015366  010066  000044          MOV     R0,44(SP)           ; *,ARR.NUM
```

```
015372  005066  000042                  CLR     42(SP)                          ; SEL.BNK                       5507
015376  000167  002740                  JMP     48$
015402  016646  000016          1$:     MOV     16(SP),-(SP)                    ; BNK.NUM,*                     5509
015406  004767  177600                  JSR     PC,BIT.CLR.OLD.NEW
015412  016646  000020                  MOV     20(SP),-(SP)                    ; BNK.NUM,*                     5511
015416  004767  173622                  JSR     PC,S.BLAST.TBL
015422  005726                          TST     (SP)+
015424  006000                          ROR     R0
015426  103402                          BLO     2$
015430  000167  002674                  JMP     47$
015434  016666  000020  000022    2$:     MOV     20(SP),22(SP)                   ; BNK.NUM,*                     5514
015442  006266  000022                  ASR     22(SP)
015446  006266  000022                  ASR     22(SP)
015452  006266  000022                  ASR     22(SP)
015456  016646  000022                  MOV     22(SP),-(SP)
015462  062716  040420'                 ADD     #BAD.BNK.REG,(SP)
015466  016646  000022                  MOV     22(SP),-(SP)                    ; BNK.NUM,*
015472  042716  177770                  BIC     #177770,(SP)
015476  012746  000001                  MOV     #1,-(SP)
015502  011646                          MOV     (SP),-(SP)
015504  004767  000000G                 JSR     PC,BL$PU2
015510  016600  000030                  MOV     30(SP),R0                       ; BNK.NUM,*                     5526
015514  000300                          SWAB    R0
015516  105000                          CLRB    R0
015520  006300                          ASL     R0
015522  010003                          MOV     R0,R3
015524  005005                          CLR     R5                              ; NIB.NUM                       5516
015526  005066  000044          3$:     CLR     44(SP)                          ; ROW.0.127.OFF                 5518
015532  005066  000046                  CLR     46(SP)                          ; ROW.128.255.OFF               5519
015536  005066  000050                  CLR     50(SP)                          ; COL.0.127.OFF                 5520
015542  005066  000052                  CLR     52(SP)                          ; COL.128.255.OFF               5521
015546  005002                          CLR     R2                              ; ROW.NUM                       5523
015550  010304          4$:     MOV     R3,R4                           ;                               5526
015552  060204                          ADD     R2,R4                           ; ROW.NUM,*
015554  006304                          ASL     R4
015556  010446                          MOV     R4,-(SP)
015560  062716  030372'                 ADD     #BLAST.TBL,(SP)
015564  010546                          MOV     R5,-(SP)                        ; NIB.NUM,*
015566  012746  000001                  MOV     #1,-(SP)
015572  005046                          CLR     -(SP)
015574  004767  000000G                 JSR     PC,BL$GT2
015600  062706  000006                  ADD     #6,SP
015604  006000                          ROR     R0
015606  005566  000046                  ADC     46(SP)                          ; ROW.0.127.OFF                 5528
015612  010416                          MOV     R4,(SP)                         ;                               5530
015614  062716  010230'                 ADD     #ERROR.MAP,(SP)
015620  010546                          MOV     R5,-(SP)                        ; NIB.NUM,*
015622  012746  000001                  MOV     #1,-(SP)
015626  005046                          CLR     -(SP)
015630  004767  000000G                 JSR     PC,BL$GT2
015634  062706  000010                  ADD     #10,SP
015640  006000                          ROR     R0
```

```
015642  005566  000044                      ADC     44(SP)                  ; ROW.0.127.OFF                 5532
015646  005202                              INC     R2                      ; ROW.NUM                       5523
015650  020227  000177                      CMP     R2,#177                 ; ROW.NUM,*
015654  003735                              BLE     4$
015656  005002                              CLR     R2                      ; COL.NUM                       5536
015660  010204                      5$:     MOV     R2,R4                   ; COL.NUM,*                     5539
015662  066704  040372'                     ADD     COL.BASE,R4
015666  060304                              ADD     R3,R4
015670  006304                              ASL     R4
015672  062704  030372'                     ADD     #BLAST.TBL,R4
015676  010446                              MOV     R4,-(SP)
015700  010546                              MOV     R5,-(SP)                ; NIB.NUM,*
015702  012746  000001                      MOV     #1,-(SP)
015706  005046                              CLR     -(SP)
015710  004767  000000G                     JSR     PC,BL$GT2
015714  062706  000006                      ADD     #6,SP
015720  006000                              ROR     R0
015722  005566  000052                      ADC     52(SP)                  ; COL.0.127.OFF                 5541
015726  010204                              MOV     R2,R4                   ; COL.NUM,*                     5543
015730  066704  040372'                     ADD     COL.BASE,R4
015734  060304                              ADD     R3,R4
015736  006304                              ASL     R4
015740  062704  010230'                     ADD     #ERROR.MAP,R4
015744  010416                              MOV     R4,(SP)
015746  010546                              MOV     R5,-(SP)                ; NIB.NUM,*
015750  012746  000001                      MOV     #1,-(SP)
015754  005046                              CLR     -(SP)
015756  004767  000000G                     JSR     PC,BL$GT2
015762  062706  000010                      ADD     #10,SP
015766  006000                              ROR     R0
015770  005566  000050                      ADC     50(SP)                  ; COL.0.127.OFF                 5545
015774  005202                              INC     R2                      ; COL.NUM                       5536
015776  020227  000177                      CMP     R2,#177                 ; COL.NUM,*
016002  003726                              BLE     5$
016004  026727  040416' 000177              CMP     BNK.NUM.SEC,#177        ;                               5549
016012  003543                              BLE     8$
016014  012702  000200                      MOV     #200,R2                 ; *,ROW.NUM                     5553
016020  010304                      6$:     MOV     R3,R4                   ;                               5559
016022  060204                              ADD     R2,R4                   ; ROW.NUM,*
016024  006304                              ASL     R4
016026  010446                              MOV     R4,-(SP)
016030  062716  030372'                     ADD     #BLAST.TBL,(SP)
016034  010546                              MOV     R5,-(SP)                ; NIB.NUM,*
016036  012746  000001                      MOV     #1,-(SP)
016042  005046                              CLR     -(SP)
016044  004767  000000G                     JSR     PC,BL$GT2
016050  062706  000006                      ADD     #6,SP
016054  006000                              ROR     R0
016056  005566  000050                      ADC     50(SP)                  ; ROW.128.255.OFF               5561
016062  010416                              MOV     R4,(SP)                 ;                               5563
016064  062716  010230'                     ADD     #ERROR.MAP,(SP)
016070  010546                              MOV     R5,-(SP)                ; NIB.NUM,*
```

```
016072  012746  000001                        MOV     #1,-(SP)
016076  005046                                CLR     -(SP)
016100  004767  000000G                       JSR     PC,BL$GT2
016104  062706  000010                        ADD     #10,SP
016110  006000                                ROR     R0
016112  005566  000046                        ADC     46(SP)                  ; ROW.128.255.OFF                       5565
016116  005202                                INC     R2                      ; ROW.NUM                               5553
016120  020227  000377                        CMP     R2,#377                 ; ROW.NUM,*
016124  003735                                BLE     6$
016126  012702  000200                        MOV     #200,R2                 ; *,COL.NUM                             5569
016132  010204                        7$:     MOV     R2,R4                   ; COL.NUM,*                             5572
016134  066704  040372'                       ADD     COL.BASE,R4
016140  060304                                ADD     R3,R4
016142  006304                                ASL     R4
016144  062704  030372'                       ADD     #BLAST.TBL,R4
016150  010446                                MOV     R4,-(SP)
016152  010546                                MOV     R5,-(SP)                ; NIB.NUM,*
016154  012746  000001                        MOV     #1,-(SP)
016160  005046                                CLR     -(SP)
016162  004767  000000G                       JSR     PC,BL$GT2
016166  062706  000006                        ADD     #6,SP
016172  006000                                ROR     R0
016174  005566  000054                        ADC     54(SP)                  ; COL.128.255.OFF                       5574
016200  010204                                MOV     R2,R4                   ; COL.NUM,*                             5576
016202  066704  040372'                       ADD     COL.BASE,R4
016206  060304                                ADD     R3,R4
016210  006304                                ASL     R4
016212  062704  010230'                       ADD     #ERROR.MAP,R4
016216  010416                                MOV     R4,(SP)
016220  010546                                MOV     R5,-(SP)                ; NIB.NUM,*
016222  012746  000001                        MOV     #1,-(SP)
016226  005046                                CLR     -(SP)
016230  004767  000000G                       JSR     PC,BL$GT2
016234  062706  000010                        ADD     #10,SP
016240  006000                                ROR     R0
016242  005566  000052                        ADC     52(SP)                  ; COL.128.255.OFF                       5578
016246  005202                                INC     R2                      ; COL.NUM                               5569
016250  020227  000377                        CMP     R2,#377                 ; COL.NUM,*
016254  003726                                BLE     7$
016256  016666  000046  000036                MOV     46(SP),36(SP)           ; ROW.128.255.OFF,QD.1.SUM              5582
016264  066666  000050  000036                ADD     50(SP),36(SP)           ; COL.0.127.OFF,QD.1.SUM
016272  016666  000044  000040                MOV     44(SP),40(SP)           ; ROW.0.127.OFF,QD.2.SUM                5583
016300  066666  000052  000040                ADD     52(SP),40(SP)           ; COL.128.255.OFF,QD.2.SUM
016306  016666  000046  000042                MOV     46(SP),42(SP)           ; ROW.128.255.OFF,QD.3.SUM              5584
016314  066666  000052  000042                ADD     52(SP),42(SP)           ; COL.128.255.OFF,QD.3.SUM
016322  016666  000044  000034        8$:     MOV     44(SP),34(SP)           ; ROW.0.127.OFF,QD.0.SUM                5587
016330  066666  000050  000034                ADD     50(SP),34(SP)           ; COL.0.127.OFF,QD.0.SUM
016336  026627  000034  000016                CMP     34(SP),#16              ; QD.0.SUM,*                            5589
016344  003571                                BLE     17$
016346  104454                                TRAP    54                      ;                                       5592
016350  000007                                .WORD   7
016352  001422'                               .WORD   CON.C.MSG
```

```
016354  000000                          .WORD   0
016356  010546                          MOV     R5,-(SP)                ; NIB.NUM,*                     5593
016360  016646  000032                  MOV     32(SP),-(SP)            ; BNK.NUM,*
016364  016646  000062                  MOV     62(SP),-(SP)            ; ARR.NUM,*
016370  012746  003124'                 MOV     #A.B.N.PRINT,-(SP)
016374  012746  000004                  MOV     #4,-(SP)
016400  010600                          MOV     SP,R0                   ; SP,*
016402  104414                          TRAP    14
016404  016616  000044                  MOV     44(SP),(SP)             ;                               5594
016410  062716  040424'                 ADD     #DEGRADE.MOD.REG,(SP)
016414  016646  000042                  MOV     42(SP),-(SP)            ; BNK.NUM,*
016420  042716  177770                  BIC     #177770,(SP)
016424  012746  000001                  MOV     #1,-(SP)
016430  011646                          MOV     (SP),-(SP)
016432  004767  000000G                 JSR     PC,BL$PU2
016436  016601  000054                  MOV     54(SP),R1               ; QD.0.SUM,OVER.FLOW            5595
016442  162701  000016                  SUB     #16,R1                  ; *,OVER.FLOW
016446  005002                          CLR     R2                      ; ROW.NUM                       5597
016450  010304                  9$:     MOV     R3,R4                   ;                               5599
016452  060204                          ADD     R2,R4                   ; ROW.NUM,*
016454  006304                          ASL     R4
016456  062704  030372'                 ADD     #BLAST.TBL,R4
016462  010446                          MOV     R4,-(SP)
016464  010546                          MOV     R5,-(SP)                ; NIB.NUM,*
016466  012746  000001                  MOV     #1,-(SP)
016472  005046                          CLR     -(SP)
016474  004767  000000G                 JSR     PC,BL$GT2
016500  062706  000010                  ADD     #10,SP
016504  006000                          ROR     R0
016506  103021                          BCC     11$
016510  010446                          MOV     R4,-(SP)                ;                               5602
016512  010546                          MOV     R5,-(SP)                ; NIB.NUM,*
016514  012746  000001                  MOV     #1,-(SP)
016520  005046                          CLR     -(SP)
016522  004767  000000G                 JSR     PC,BL$PU2
016526  005301                          DEC     R1                      ; OVER.FLOW                     5603
016530  005366  000070                  DEC     70(SP)                  ; QD.2.SUM                      5604
016534  005701                          TST     R1                      ; OVER.FLOW                     5606
016536  001003                          BNE     10$
016540  062706  000010                  ADD     #10,SP
016544  000406                          BR      12$
016546  062706  000010          10$:    ADD     #10,SP                  ;                               5601
016552  005202                  11$:    INC     R2                      ; ROW.NUM                       5597
016554  020227  000177                  CMP     R2,#177                 ; ROW.NUM,*
016560  003733                          BLE     9$
016562  005701                  12$:    TST     R1                      ; OVER.FLOW                     5610
016564  001457                          BEQ     16$
016566  005002                          CLR     R2                      ; COL.NUM                       5614
016570  010204                  13$:    MOV     R2,R4                   ; COL.NUM,*                     5616
016572  066704  040372'                 ADD     COL.BASE,R4
016576  060304                          ADD     R3,R4
016600  006304                          ASL     R4
```

```
016602  062704  030372'                 ADD     #BLAST.TBL,R4
016606  010446                          MOV     R4,-(SP)
016610  010546                          MOV     R5,-(SP)                ; NIB.NUM,*
016612  012746  000001                  MOV     #1,-(SP)
016616  005046                          CLR     -(SP)
016620  004767  000000G                 JSR     PC,BL$GT2
016624  062706  000010                  ADD     #10,SP
016630  006000                          ROR     R0
016632  103030                          BCC     15$
016634  010204                          MOV     R2,R4                   ; COL.NUM,*                     5619
016636  066704  040372'                 ADD     COL.BASE,R4
016642  060304                          ADD     R3,R4
016644  006304                          ASL     R4
016646  062704  030372'                 ADD     #BLAST.TBL,R4
016652  010446                          MOV     R4,-(SP)
016654  010546                          MOV     R5,-(SP)                ; NIB.NUM,*
016656  012746  000001                  MOV     #1,-(SP)
016662  005046                          CLR     -(SP)
016664  004767  000000G                 JSR     PC,BL$PU2
016670  005301                          DEC     R1                      ; OVER.FLOW                     5620
016672  005366  000066                  DEC     66(SP)                  ; QD.1.SUM                      5621
016676  005701                          TST     R1                      ; OVER.FLOW                     5623
016700  001003                          BNE     14$
016702  062706  000010                  ADD     #10,SP
016706  000406                          BR      16$
016710  062706  000010          14$:    ADD     #10,SP                  ;                               5618
016714  005202                  15$:    INC     R2                      ; COL.NUM                       5614
016716  020227  000177                  CMP     R2,#177                 ; COL.NUM,*
016722  003722                          BLE     13$
016724  062706  000020          16$:    ADD     #20,SP                  ;                               5591
016730  026727  040416'  000177  17$:    CMP     BNK.NUM.SEC,#177        ;                               5631
016736  003002                          BGT     18$
016740  000167  001334                  JMP     45$
016744  026627  000036  000016  18$:    CMP     36(SP),#16              ; QD.1.SUM,*                    5635
016752  003567                          BLE     27$
016754  104454                          TRAP    54                      ;                               5638
016756  000007                          .WORD   7
016760  001422'                         .WORD   CON.C.MSG
016762  000000                          .WORD   0
016764  010546                          MOV     R5,-(SP)                ; NIB.NUM,*                     5639
016766  016646  000032                  MOV     32(SP),-(SP)            ; BNK.NUM,*
016772  016646  000062                  MOV     62(SP),-(SP)            ; ARR.NUM,*
016776  012746  003124'                 MOV     #A.B.N.PRINT,-(SP)
017002  012746  000004                  MOV     #4,-(SP)
017006  010600                          MOV     SP,R0                   ; SP,*
017010  104414                          TRAP    14
017012  016616  000044                  MOV     44(SP),(SP)             ;                               5640
017016  062716  040424'                 ADD     #DEGRADE.MOD.REG,(SP)
017022  016646  000042                  MOV     42(SP),-(SP)            ; BNK.NUM,*
017026  042716  177770                  BIC     #177770,(SP)
017032  012746  000001                  MOV     #1,-(SP)
017036  011646                          MOV     (SP),-(SP)
```

```
017040  004767  000000G                 JSR     PC,BL$PU2
017044  016601  000056                  MOV     56(SP),R1               ; QD.1.SUM,OVER.FLOW            5641
017050  162701  000016                  SUB     #16,R1                  ; *,OVER.FLOW
017054  012702  000200                  MOV     #200,R2                 ; *,ROW.NUM                     5643
017060  010304                  19$:    MOV     R3,R4                   ;                               5645
017062  060204                          ADD     R2,R4                   ; ROW.NUM,*
017064  006304                          ASL     R4
017066  062704  030372'                  ADD     #BLAST.TBL,R4
017072  010446                          MOV     R4,-(SP)
017074  010546                          MOV     R5,-(SP)                ; NIB.NUM,*
017076  012746  000001                  MOV     #1,-(SP)
017102  005046                          CLR     -(SP)
017104  004767  000000G                 JSR     PC,BL$GT2
017110  062706  000010                  ADD     #10,SP
017114  006000                          ROR     R0
017116  103021                          BCC     21$
017120  010446                          MOV     R4,-(SP)                ;                               5648
017122  010546                          MOV     R5,-(SP)                ; NIB.NUM,*
017124  012746  000001                  MOV     #1,-(SP)
017130  005046                          CLR     -(SP)
017132  004767  000000G                 JSR     PC,BL$PU2
017136  005301                          DEC     R1                      ; OVER.FLOW                     5649
017140  005366  000072                  DEC     72(SP)                  ; QD.3.SUM                      5650
017144  005701                          TST     R1                      ; OVER.FLOW                     5652
017146  001003                          BNE     20$
017150  062706  000010                  ADD     #10,SP
017154  000406                          BR      22$
017156  062706  000010          20$:    ADD     #10,SP                  ;                               5647
017162  005202                  21$:    INC     R2                      ; ROW.NUM                       5643
017164  020227  000377                  CMP     R2,#377                 ; ROW.NUM,*
017170  003733                          BLE     19$
017172  005701                  22$:    TST     R1                      ; OVER.FLOW                     5656
017174  001454                          BEQ     26$
017176  005002                          CLR     R2                      ; COL.NUM                       5660
017200  010204                  23$:    MOV     R2,R4                   ; COL.NUM,*                     5662
017202  066704  040372'                  ADD     COL.BASE,R4
017206  060304                          ADD     R3,R4
017210  006304                          ASL     R4
017212  062704  030372'                  ADD     #BLAST.TBL,R4
017216  010446                          MOV     R4,-(SP)
017220  010546                          MOV     R5,-(SP)                ; NIB.NUM,*
017222  012746  000001                  MOV     #1,-(SP)
017226  005046                          CLR     -(SP)
017230  004767  000000G                 JSR     PC,BL$GT2
017234  062706  000010                  ADD     #10,SP
017240  006000                          ROR     R0
017242  103025                          BCC     25$
017244  010204                          MOV     R2,R4                   ; COL.NUM,*                     5665
017246  066704  040372'                  ADD     COL.BASE,R4
017252  060304                          ADD     R3,R4
017254  006304                          ASL     R4
017256  062704  030372'                  ADD     #BLAST.TBL,R4
```

```
017262  010446                          MOV     R4,-(SP)
017264  010546                          MOV     R5,-(SP)                ; NIB.NUM,*
017266  012746  000001                  MOV     #1,-(SP)
017272  005046                          CLR     -(SP)
017274  004767  000000G                 JSR     PC,BL$PU2
017300  005301                          DEC     R1                      ; OVER.FLOW                     5666
017302  001003                          BNE     24$                     ;                               5668
017304  062706  000010                  ADD     #10,SP
017310  000406                          BR      26$
017312  062706  000010          24$:    ADD     #10,SP                  ;                               5664
017316  005202          25$:    INC     R2                              ; COL.NUM                       5660
017320  020227  000177                  CMP     R2,#177                 ; COL.NUM,*
017324  003725                          BLE     23$
017326  062706  000020          26$:    ADD     #20,SP                  ;                               5637
017332  026627  000040  000016  27$:    CMP     40(SP),#16              ; QD.2.SUM,*                    5676
017340  003566                          BLE     36$
017342  104454                          TRAP    54                      ;                               5679
017344  000007                          .WORD   7
017346  001422'                         .WORD   CON.C.MSG
017350  000000                          .WORD   0
017352  010546                          MOV     R5,-(SP)                ; NIB.NUM,*                     5680
017354  016646  000032                  MOV     32(SP),-(SP)            ; BNK.NUM,*
017360  016646  000062                  MOV     62(SP),-(SP)            ; ARR.NUM,*
017364  012746  003124'                 MOV     #A.B.N.PRINT,-(SP)
017370  012746  000004                  MOV     #4,-(SP)
017374  010600                          MOV     SP,R0                   ; SP,*
017376  104414                          TRAP    14
017400  016616  000044                  MOV     44(SP),(SP)             ;                               5681
017404  062716  040424'                 ADD     #DEGRADE.MOD.REG,(SP)
017410  016646  000042                  MOV     42(SP),-(SP)            ; BNK.NUM,*
017414  042716  177770                  BIC     #177770,(SP)
017420  012746  000001                  MOV     #1,-(SP)
017424  011646                          MOV     (SP),-(SP)
017426  004767  000000G                 JSR     PC,BL$PU2
017432  016601  000060                  MOV     60(SP),R1               ; QD.2.SUM,OVER.FLOW            5682
017436  162701  000016                  SUB     #16,R1                  ; *,OVER.FLOW
017442  005002                          CLR     R2                      ; ROW.NUM                       5684
017444  010304          28$:    MOV     R3,R4                           ;                               5686
017446  060204                          ADD     R2,R4                   ; ROW.NUM,*
017450  006304                          ASL     R4
017452  062704  030372'                 ADD     #BLAST.TBL,R4
017456  010446                          MOV     R4,-(SP)
017460  010546                          MOV     R5,-(SP)                ; NIB.NUM,*
017462  012746  000001                  MOV     #1,-(SP)
017466  005046                          CLR     -(SP)
017470  004767  000000G                 JSR     PC,BL$GT2
017474  062706  000010                  ADD     #10,SP
017500  006000                          ROR     R0
017502  103016                          BCC     30$
017504  010446                          MOV     R4,-(SP)                ;                               5689
017506  010546                          MOV     R5,-(SP)                ; NIB.NUM,*
017510  012746  000001                  MOV     #1,-(SP)
```

```
017514  005046                          CLR     -(SP)
017516  004767  000000G                 JSR     PC,BL$PU2
017522  005301                          DEC     R1                      ; OVER.FLOW                     5690
017524  001003                          BNE     29$                     ;                               5692
017526  062706  000010                  ADD     #10,SP
017532  000406                          BR      31$
017534  062706  000010          29$:    ADD     #10,SP                  ;                               5688
017540  005202                  30$:    INC     R2                      ; ROW.NUM                       5684
017542  020227  000177                  CMP     R2,#177                 ; ROW.NUM,*
017546  003736                          BLE     28$
017550  005701                  31$:    TST     R1                      ; OVER.FLOW                     5696
017552  001457                          BEQ     35$
017554  012702  000200                  MOV     #200,R2                 ; *,COL.NUM                     5700
017560  010204                  32$:    MOV     R2,R4                   ; COL.NUM,*                     5702
017562  066704  040372'                 ADD     COL.BASE,R4
017566  060304                          ADD     R3,R4
017570  006304                          ASL     R4
017572  062704  030372'                 ADD     #BLAST.TBL,R4
017576  010446                          MOV     R4,-(SP)
017600  010546                          MOV     R5,-(SP)                ; NIB.NUM,*
017602  012746  000001                  MOV     #1,-(SP)
017606  005046                          CLR     -(SP)
017610  004767  000000G                 JSR     PC,BL$GT2
017614  062706  000010                  ADD     #10,SP
017620  006000                          ROR     R0
017622  103027                          BCC     34$
017624  010204                          MOV     R2,R4                   ; COL.NUM,*                     5705
017626  066704  040372'                 ADD     COL.BASE,R4
017632  060304                          ADD     R3,R4
017634  006304                          ASL     R4
017636  062704  030372'                 ADD     #BLAST.TBL,R4
017642  010446                          MOV     R4,-(SP)
017644  010546                          MOV     R5,-(SP)                ; NIB.NUM,*
017646  012746  000001                  MOV     #1,-(SP)
017652  005046                          CLR     -(SP)
017654  004767  000000G                 JSR     PC,BL$PU2
017660  005366  000072                  DEC     72(SP)                  ; QD.3.SUM                      5706
017664  005301                          DEC     R1                      ; OVER.FLOW                     5707
017666  001003                          BNE     33$                     ;                               5709
017670  062706  000010                  ADD     #10,SP
017674  000406                          BR      35$
017676  062706  000010          33$:    ADD     #10,SP                  ;                               5704
017702  005202                  34$:    INC     R2                      ; COL.NUM                       5700
017704  020227  000377                  CMP     R2,#377                 ; COL.NUM,*
017710  003723                          BLE     32$
017712  062706  000020          35$:    ADD     #20,SP                  ;                               5678
017716  026627  000042  000016  36$:    CMP     42(SP),#16              ; QD.3.SUM,*                    5717
017724  003565                          BLE     45$
017726  104454                          TRAP    54                      ;                               5720
017730  000007                          .WORD   7
017732  001422'                         .WORD   CON.C.MSG
017734  000000                          .WORD   0
```

```
017736  010546                          MOV     R5,-(SP)                ; NIB.NUM,*                     5721
017740  016646  000032                  MOV     32(SP),-(SP)            ; BNK.NUM,*
017744  016646  000062                  MOV     62(SP),-(SP)            ; ARR.NUM,*
017750  012746  003124'                 MOV     #A.B.N.PRINT,-(SP)
017754  012746  000004                  MOV     #4,-(SP)
017760  010600                          MOV     SP,R0                   ; SP,*
017762  104414                          TRAP    14
017764  016616  000044                  MOV     44(SP),(SP)             ;                               5722
017770  062716  040424'                 ADD     #DEGRADE.MOD.REG,(SP)
017774  016646  000042                  MOV     42(SP),-(SP)            ; BNK.NUM,*
020000  042716  177770                  BIC     #177770,(SP)
020004  012746  000001                  MOV     #1,-(SP)
020010  011646                          MOV     (SP),-(SP)
020012  004767  000000G                 JSR     PC,BL$PU2
020016  016601  000062                  MOV     62(SP),R1               ; QD.3.SUM,OVER.FLOW            5723
020022  162701  000016                  SUB     #16,R1                  ; *,OVER.FLOW
020026  012702  000200                  MOV     #200,R2                 ; *,ROW.NUM                     5725
020032  010304                  37$:    MOV     R3,R4                   ;                               5727
020034  060204                          ADD     R2,R4                   ; ROW.NUM,*
020036  006304                          ASL     R4
020040  062704  030372'                 ADD     #BLAST.TBL,R4
020044  010446                          MOV     R4,-(SP)
020046  010546                          MOV     R5,-(SP)                ; NIB.NUM,*
020050  012746  000001                  MOV     #1,-(SP)
020054  005046                          CLR     -(SP)
020056  004767  000000G                 JSR     PC,BL$GT2
020062  062706  000010                  ADD     #10,SP
020066  006000                          ROR     R0
020070  103016                          BCC     39$
020072  010446                          MOV     R4,-(SP)                ;                               5730
020074  010546                          MOV     R5,-(SP)                ; NIB.NUM,*
020076  012746  000001                  MOV     #1,-(SP)
020102  005046                          CLR     -(SP)
020104  004767  000000G                 JSR     PC,BL$PU2
020110  005301                          DEC     R1                      ; OVER.FLOW                     5731
020112  001003                          BNE     38$                     ;                               5733
020114  062706  000010                  ADD     #10,SP
020120  000406                          BR      40$
020122  062706  000010          38$:    ADD     #10,SP                  ;                               5729
020126  005202                  39$:    INC     R2                      ; ROW.NUM                       5725
020130  020227  000377                  CMP     R2,#377                 ; ROW.NUM,*
020134  003736                          BLE     37$
020136  005701                  40$:    TST     R1                      ; OVER.FLOW                     5737
020140  001455                          BEQ     44$
020142  012702  000200                  MOV     #200,R2                 ; *,COL.NUM                     5741
020146  010204                  41$:    MOV     R2,R4                   ; COL.NUM,*                     5743
020150  066704  040372'                 ADD     COL.BASE,R4
020154  060304                          ADD     R3,R4
020156  006304                          ASL     R4
020160  062704  030372'                 ADD     #BLAST.TBL,R4
020164  010446                          MOV     R4,-(SP)
020166  010546                          MOV     R5,-(SP)                ; NIB.NUM,*
```

```
020170  012746  000001                  MOV     #1,-(SP)
020174  005046                          CLR     -(SP)
020176  004767  000000G                 JSR     PC,BL$GT2
020202  062706  000010                  ADD     #10,SP
020206  006000                          ROR     R0
020210  103025                          BCC     43$
020212  010204                          MOV     R2,R4                   ; COL.NUM,*                     5746
020214  066704  040372'                 ADD     COL.BASE,R4
020220  060304                          ADD     R3,R4
020222  006304                          ASL     R4
020224  062704  030372'                 ADD     #BLAST.TBL,R4
020230  010446                          MOV     R4,-(SP)
020232  010546                          MOV     R5,-(SP)                ; NIB.NUM,*
020234  012746  000001                  MOV     #1,-(SP)
020240  005046                          CLR     -(SP)
020242  004767  000000G                 JSR     PC,BL$PU2
020246  005301                          DEC     R1                      ; OVER.FLOW                     5747
020250  001003                          BNE     42$                     ;                               5749
020252  062706  000010                  ADD     #10,SP
020256  000406                          BR      44$
020260  062706  000010          42$:    ADD     #10,SP                  ;                               5745
020264  005202                  43$:    INC     R2                      ; COL.NUM                       5741
020266  020227  000377                  CMP     R2,#377                 ; COL.NUM,*
020272  003725                          BLE     41$
020274  062706  000020          44$:    ADD     #20,SP                  ;                               5719
020300  005205                  45$:    INC     R5                      ; NIB.NUM                       5516
020302  020527  000011                  CMP     R5,#11                  ; NIB.NUM,*
020306  003002                          BGT     46$
020310  000167  175212                  JMP     3$
020314  016616  000030          46$:    MOV     30(SP),(SP)             ; BNK.NUM,*                     5761
020320  004767  174600                  JSR     PC,OR.OLD.NEW.PD
020324  062706  000010                  ADD     #10,SP                  ;                               5513
020330  005266  000020          47$:    INC     20(SP)                  ; BNK.NUM                       5764
020334  005726                          TST     (SP)+                   ;                               5508
020336  005266  000042                  INC     42(SP)                  ; SEL.BNK                       5507
020342  026666  000042  000064  48$:    CMP     42(SP),64(SP)           ; SEL.BNK,TSTED.BNK
020350  002002                          BGE     49$
020352  000167  175024                  JMP     1$
020356  062706  000046          49$:    ADD     #46,SP                  ;                               5427
020362  000207                          RTS     PC

; Routine Size:  799 words
; Maximum stack depth per invocation:  42 words
```

```
;          routine IN_ERROR_MAP (TEMP_ARR$BNK_SEL, BAD_CHIP) : novalue =
;              begin
;
;          !++
;          ! Functional Description:
;          !       The error map is loaded with all the new and
;          !       old failing row and sector addresses for a
;          !       failing chip.
;          !
;          !       From this row and sector information stored
;          !       here it can be calculated which are newly
;          !       failing rows and columns and the best way
;          !       to mask them.
;          !
;          !       This global routine will interigate the error
;          !       map and load the blast table with this chips
;          !       new masking prom data.
;          !
;          !       This global routine executes in the following
;          !       steps:
;          !         1. It first searches the error map for
;          !            all bad rows IE. occurances of bad
;          !            rows > 10 times.
;          !
;          !         2. It then searches the column count
;          !            table for column counts > 10
;          !
;          !         3. It then transfers to the remainder table
;          !            any remaining single cell failures.
;          !
;          !         4. It then interigates the remainder table
;          !            for these remaining single cell failures
;          !            and selects the best one for blasting.
;          !
;          ! Formal Parameters:
;          !       ARR$BNK_SEL:
;          !               Stores array and bank select address.
;          !       BAD_CHIP:
;          !               Points to the failing chip persently
;          !               being pm'ed.
;          !
;          ! Implicit Inputs:
;          !       DEGRADE_MOD_REG, FLG_REG
;          !
;          ! Implicit Outputs:
;          !       If it is determined that this is the first
;          !       all bad chip in this bank then the flag register
;          !       and degrade register bits are set.
;          !
;          !       If this is the second all bad chip in this bank
;          !       then the flag register bit abort_array is set.
;          !
```

```
;          5820  ! Completion codes:
;          5821  !       none
;          5822  !
;          5823  ! Side Effects:
;          5824  !       none
;          5825  !--
;          5826
;          5827      local
;          5828          ARR_SEL,                                !Stores array select number
;          5829          BNK_NUM,                                !Stores bank sel number
;          5830          ALL_BAD;                                !Records total all bad rows and columns found bad
;          5831
;          5832      ARR_SEL = .TEMP_ARR$BNK_SEL<.ARR_SEL_POS, ARR$SEL_SIZE>;    !Load the array sel number
;          5833      BNK_NUM = .TEMP_ARR$BNK_SEL<.BNK_SEL_POS, BNK$SEL_SIZE>;    !Load the bnk sel number
;          5834      I_TMP_BLST_TBL ();                          !Init the temporary blast table
;          5835      !
;          5836      ! Search the error map and find failing row
;          5837      ! addresses which have failed more than 10
;          5838      ! times.  For the rows that don't fail more
;          5839      ! than ten times save a count of the adjacent
;          5840      ! column pair which will later be used to find
;          5841      ! these rows which failed less than 10 times.
;          5842      !
;          5843      ALL_BAD = S_ERROR_MAP (.BAD_CHIP);          !Search error map for all bad rows
;          5844      !
;          5845      ! Did the search of the error map find 10 all
;          5846      ! bad rows in error within this chip ?
;          5847      !
;          5848      ! If 10 were found then this is called an all
;          5849      ! bad chip else continue and search the column
;          5850      ! count table for columns in error greater than
;          5851      ! 10 times.
;          5852      !
;          5853
;          5854      if .ALL_BAD leq 9                           !Is the all bad count <= 9
;          5855      then
;          5856          begin
;          5857          !
;          5858          ! Search the column count table for column counts
;          5859          ! greater than ten.
;          5860          !
;          5861          ALL_BAD = S_COL_CNT_TBL (.BAD_CHIP, .ALL_BAD);  !Search column count table for all bad cols
;          5862          !
;          5863          ! If the search error map plus search column count table
;          5864          ! found 10 all bad rows and columns then this chip is called
;          5865          ! all bad and no prom blasting is done else search the remainder
;          5866          ! table for remaining row column pairs for blasting.
;          5867          !
;          5868
;          5869          if .ALL_BAD leq 9                       !Is the all bad count <= 9
;          5870          then
;          5871              begin
```

```
5872                ALL_BAD = S_REM_TBL (.BAD_CHIP, .ALL_BAD);  !Find the remaining bad rows/cols
5873                end;
5874
5875            end;
5876
5877        !
5878        ! For the purpose of report code summary
5879        ! store into a statistical report table
5880        ! the locations, chip numbers and the number
5881        ! of bad rows and columns blasted in each newly
5882        ! failing mos ram chip.
5883        !
5884
5885        if .PM_COUNT lss 128                            !Only store a total of 128 failing chips
5886        then
5887            begin
5888            PM_LOG [.PM_COUNT, BIT$_PM] = .BAD_CHIP;        !Store the chip number
5889            PM_LOG [.PM_COUNT, BNK$_PM] = .BNK_NUM; !Store chips bank number
5890            PM_LOG [.PM_COUNT, BRD$_PM] = .ARR_SEL; !Store chips array numbrer
5891            PM_LOG [.PM_COUNT, UNIT$_PM] = .ML_DUT; !Store chips unit number
5892            PM_LOG [.PM_COUNT, SUM$_PM] = .ALL_BAD + 1;     !Store number of r/c failing
5893            PM_COUNT = .PM_COUNT + 1;                   !Up the number of table enters
5894            end;
5895
5896        if .ALL_BAD leq 9                               !Were there <= 9 all bad rows/cols found
5897        then
5898            begin
5899            X_TMP_BLST_TBL (.TEMP_ARR$BNK_SEL, .ALL_BAD);   !Xfer temp blast table to blast table
5900            end
5901        else
5902            begin
5903
5904            if not .FLG_REG [F_ALL_BAD_CHIP]            !Is this the first all bad chip in this bank
5905            then
5906                begin
5907                ERRSF (ERR_5, CON_A_MSG, 0);            !Report a condition a
5908                PRINTB (A_B_C_PRINT, .ARR_SEL, .BNK_NUM, .BAD_CHIP);        !Report where
5909                FLG_REG [F_ALL_BAD_CHIP] = SET_FLG; !Set the all bad chip flag
5910                DEGRADE_MOD_REG [.BNK_NUM] = SET_FLG;
5911                end
5912            else
5913                begin
5914                ERRSF (ERR_6, CON_B_MSG, 0);            !Report a condition b
5915                PRINTB (REP_PRINT, .ARR_SEL);           !Report where
5916                FLG_REG [F_ABORT_ARRAY] = SET_FLG;  !Set the abort array flag
5917                end;
5918
5919            end;
5920
5921        end;
```

```
                                        .SBTTL  IN.ERROR.MAP ROUTINE DECLARATIONS
020364  004167  000000G        IN.ERROR.MAP:
                                        JSR     R1,$SAVE5                       ;                               5768
020370  012746  000022                  MOV     #22,-(SP)                       ;                               5832
020374  060616                          ADD     SP,(SP)                         ; TEMP.ARR$BNK.SE,*
020376  016746  040412'                  MOV     ARR.SEL.POS,-(SP)
020402  012746  000004                  MOV     #4,-(SP)
020406  005046                          CLR     -(SP)
020410  004767  000000G                  JSR     PC,BL$GT2
020414  010005                          MOV     R0,R5                           ; *,ARR.SEL
020416  012716  000030                  MOV     #30,(SP)                        ;                               5833
020422  060616                          ADD     SP,(SP)                         ; TEMP.ARR$BNK.SE,*
020424  016746  040414'                  MOV     BNK.SEL.POS,-(SP)
020430  012746  000002                  MOV     #2,-(SP)
020434  005046                          CLR     -(SP)
020436  004767  000000G                  JSR     PC,BL$GT2
020442  010001                          MOV     R0,R1                           ; *,BNK.NUM
020444  004767  164460                  JSR     PC,I.TMP.BLST.TBL               ;                               5834
020450  016603  000034                  MOV     34(SP),R3                       ; BAD.CHIP,*                    5843
020454  010316                          MOV     R3,(SP)
020456  004767  172000                  JSR     PC,S.ERROR.MAP
020462  020027  000011                  CMP     R0,#11                          ; ALL.BAD,*                     5854
020466  003015                          BGT     2$
020470  010346                          MOV     R3,-(SP)                        ;                               5861
020472  010046                          MOV     R0,-(SP)                        ; ALL.BAD,*
020474  004767  171610                  JSR     PC,S.COL.CNT.TBL
020500  020027  000011                  CMP     R0,#11                          ; ALL.BAD,*                     5869
020504  003005                          BGT     1$
020506  010346                          MOV     R3,-(SP)                        ;                               5872
020510  010046                          MOV     R0,-(SP)                        ; ALL.BAD,*
020512  004767  171216                  JSR     PC,S.REM.TBL
020516  022626                          CMP     (SP)+,(SP)+                     ;                               5871
020520  022626                  1$:     CMP     (SP)+,(SP)+                     ;                               5856
020522  026727  000000'  000200  2$:     CMP     PM.COUNT,#200                   ;                               5885
020530  002060                          BGE     3$
020532  016704  000000'                  MOV     PM.COUNT,R4                     ;                               5888
020536  006304                          ASL     R4
020540  006304                          ASL     R4
020542  012702  000002'                  MOV     #PM.LOG,R2
020546  060402                          ADD     R4,R2
020550  010346                          MOV     R3,-(SP)
020552  000316                          SWAB    (SP)
020554  106016                          RORB    (SP)
020556  006016                          ROR     (SP)
020560  006016                          ROR     (SP)
020562  042716  160077                  BIC     #160077,(SP)
020566  042712  017700                  BIC     #17700,(R2)
020572  052612                          BIS     (SP)+,(R2)
020574  010146                          MOV     R1,-(SP)                        ; BNK.NUM,*                     5889
020576  042716  177774                  BIC     #177774,(SP)
020602  142712  000003                  BICB    #3,(R2)
020606  152612                          BISB    (SP)+,(R2)
```

```
020610  010546                          MOV     R5,-(SP)                ; ARR.SEL,*                     5890
020612  006316                          ASL     (SP)
020614  006316                          ASL     (SP)
020616  042716  177703                  BIC     #177703,(SP)
020622  142712  000074                  BICB    #74,(R2)
020626  152612                          BISB    (SP)+,(R2)
020630  016745  040400'                  MOV     ML.DUT,-(SP)            ;                               5891
020634  006016                          ROR     (SP)
020636  006016                          ROR     (SP)
020640  006016                          ROR     (SP)
020642  006016                          ROR     (SP)
020644  042716  017777                  BIC     #17777,(SP)
020650  042712  160000                  BIC     #160000,(R2)
020654  052612                          BIS     (SP)+,(R2)
020656  010064  000004'                  MOV     R0,PM.LOG+2(R4)         ; ALL.BAD,*                     5892
020662  005264  000004'                  INC     PM.LOG+2(R4)
020666  005267  000000'                  INC     PM.COUNT                ;                               5893
020672  020027  000011          3$:     CMP     R0,#11                  ; ALL.BAD,*                     5896
020676  003006                          BGT     4$
020700  016646  000036                  MOV     36(SP),-(SP)            ; TEMP.ARR$BNK.SE,*             5899
020704  010046                          MOV     R0,-(SP)                ; ALL.BAD,*
020706  004767  166512                  JSR     PC,X.TMP.BLST.TBL
020712  000465                          BR      7$                      ;                               5896
020714  032767  000020  040422'  4$:     BIT     #20,FLG.REG             ;                               5904
020722  001042                          BNE     5$
020724  104454                          TRAP    54                      ;                               5907
020726  000005                          .WORD   5
020730  001372'                         .WORD   CON.A.MSG
020732  000000                          .WORD   0
020734  010346                          MOV     R3,-(SP)                ;                               5908
020736  010146                          MOV     R1,-(SP)                ; BNK.NUM,*
020740  010546                          MOV     R5,-(SP)                ; ARR.SEL,*
020742  012746  003052'                  MOV     #A.B.C.PRINT,-(SP)
020746  012746  000004                  MOV     #4,-(SP)
020752  010600                          MOV     SP,R0                   ; SP,*
020754  104414                          TRAP    14
020756  152767  000020  040422'          BISB    #20,FLG.REG             ;                               5909
020764  010100                          MOV     R1,R0                   ; BNK.NUM,*                     5910
020766  006200                          ASR     R0
020770  006200                          ASR     R0
020772  006200                          ASR     R0
020774  062700  040424'                  ADD     #DEGRADE.MOD.REG,R0
021000  010016                          MOV     R0,(SP)
021002  010146                          MOV     R1,-(SP)                ; BNK.NUM,*
021004  042716  177770                  BIC     #177770,(SP)
021010  012746  000001                  MOV     #1,-(SP)
021014  011646                          MOV     (SP),-(SP)
021016  004767  000000G                 JSR     PC,BL$PU2
021022  062706  000012                  ADD     #12,SP                  ;                               5906
021026  000416                          BR      6$                      ;                               5904
021030  104454                  5$:     TRAP    54                      ;                               5914
021032  000006                          .WORD   6
```

```
021034  001406'                         .WORD   CON.B.MSG
021036  000000                          .WORD   0
021040  010546                          MOV     R5,-(SP)                ; ARR.SEL,*                             5915
021042  012746  003200'                 MOV     #REP.PRINT,-(SP)
021046  012746  000002                  MOV     #2,-(SP)
021052  010600                          MOV     SP,R0                   ; SP,*
021054  104414                          TRAP    14
021056  152767  000100  040422'         BISB    #100,FLG.REG            ;                                       5916
021064  005726                  6$:     TST     (SP)+                   ;                                       5902
021066  062706  000022          7$:     ADD     #22,SP                  ;                                       5769
021072  000207                          RTS     PC                      ;                                       5768

; Routine Size:  164 words
; Maximum stack depth per invocation:  21 words
```

```
;         5922  routine FB_DATA_CHIPS (TEMP_ARR$BNK_SEL) : novalue =
;         5923      begin
;         5924
;         5925  !++
;         5926  ! Functional Description:
;         5927  !       This global routine will search the selected banks
;         5928  !       via mass bus transfers looking for additional
;         5929  !       failing chips.
;         5930  !
;         5931  !       If a bad chip is detected then the bit
;         5932  !       from which the failure came from is calculated
;         5933  !       and the failing chip number and pattern is
;         5934  !       stored into the bad chip table.
;         5935  !
;         5936  ! Formal Parameters:
;         5937  !       ARR$BNK_SEL:
;         5938  !               Stores array and bank select address.
;         5939  !
;         5940  ! Implicit Inputs:
;         5941  !       RET_STATUS, WRT_BUF, RD_BUF, FLG_REG
;         5942  !
;         5943  ! Implicit Outputs:
;         5944  !       The flag register bits will be set if the write
;         5945  !       check transfers detects an UNC error.
;         5946  !
;         5947  ! Completion codes:
;         5948  !       none
;         5949  !
;         5950  ! Side Effects:
;         5951  !       none
;         5952  !--
;         5953
;         5954      local
;         5955          BITS_TSTED,                         !Counts no. bits tested in read buffer
;         5956          OFFSET,                             !Random data offset
;         5957          RAND_INDEX,                         !Starts sector transfer at random write buffer words
;         5958          BAD_BITS : bitvector [16],          !Loaded with failing read buffer word
;         5959          FAIL_CHIP;                          !Records the failing ml-11 data chip
;         5960
;         5961      SEED1 = ZEROES;                         !Init random data seed1
;         5962      SEED2 = %o'1233';                       !Init random data seed2
;         5963      SEED3 = %o'7622';                       !Init random data seed3
;         5964      OFFSET = ZEROES;                        !Clear the random data offset for 1's and 0's data
;         5965
;         5966      incr PAT_SEL from 0 to ONE + .RAND_PASS do  !Loop thru all patterns
;         5967          begin
;         5968          RAND_INDEX = ZEROES;                !Reset random index to zero
;         5969
;         5970          selectone .PAT_SEL of               !Select a pattern
;         5971              set
;         5972
;         5973              [0] :
```

```
5974                    L_1_0_DATA (.TEMP_ARR$BNK_SEL, ZEROES); !Load zeroes data
5975
5976                [1] :
5977                    L_1_0_DATA (.TEMP_ARR$BNK_SEL, ONES);   !Load ones data
5978
5979                [otherwise] :
5980                    begin
5981                    L_RANDOM_DATA (.TEMP_ARR$BNK_SEL);      !Load random data
5982                    OFFSET = 2;                             !Enable the random data offset
5983                    end;
5984                tes;
5985
5986            incr SEC_CNT from 0 to .BNK_NUM_SEC do  !Write check all sectors in this bank
5987                begin
5988                BREAK;
5989                WRT_CHK_TRANSFER (SIZE = -256, DST = (.TEMP_ARR$BNK_SEL + .SEC_CNT),
5990                    SRC = WRT_BUF [.RAND_INDEX, WRD]);
5991
5992                if .ML_ADDR [MLEE, crc]                 !Was there a crc bit error
5993                then
5994                    begin
5995                    FAIL_CHIP = .ML_ADDR [MLEE, CHAN];      !Read the failing crc bit number
5996                    L_CHIP_TBL (.FAIL_CHIP, .PAT_SEL);      !Store away the failing bit and pattern
5997                    end
5998                else
5999                    begin
6000
6001                    if .ML_ADDR [MLEE, UNC]             !Was there a uncorrectable error
6002                    then
6003                        begin
6004                        FLG_REG [F_UNC_ERR_FLG] = SET_FLG;  !Set the unc error flag for future ref
6005                        end;
6006
6007                    end;
6008
6009                if .ML_ADDR [MLCS2, WCE]                !Was there a write check error
6010                then
6011                    begin
6012                    CLR_MBUS;                           !Clear the mass bus of the error
6013                    BITS_TSTED = ZEROES;                !Clear the bits tested count
6014                    RD_TRANSFER (SIZE = -256, DST = RD_BUF, SRC = .TEMP_ARR$BNK_SEL + .SEC_CNT);
6015
6016                    incr BUF_INDEX from 0 to 255 do         !Find the failing read buffer word
6017                        begin
6018
6019                        if .WRT_BUF [(.RAND_INDEX + .BUF_INDEX), WRD] neq .RD_BUF [.BUF_INDEX]
6020                        then
6021                            begin
6022                            BAD_BITS = .WRT_BUF [(.RAND_INDEX + .BUF_INDEX), WRD] xor .RD_BUF [.BUF_INDEX];
6023
6024                            incr BIT_INDEX from 0 to 15 do  !Find the failing bit(s)
6025
```

```
;         6026                        if .BAD_BITS [.BIT_INDEX] eql ONE   !Is this a bad bit
;         6027                        then
;         6028                            begin
;         6029                            FAIL_CHIP = (.BITS_TSTED + .BIT_INDEX) mod 36;
;         6030                                        !Calculate the failing ml-11 bit
;         6031                            L_CHIP_TBL (.FAIL_CHIP, .PAT_SEL);      !Store away the bit and pattern
;         6032                            end;
;         6033
;         6034                        end;
;         6035
;         6036                    BITS_TSTED = .BITS_TSTED + 16;      !Add 16 bit to the bit tested count
;         6037                    end;
;         6038
;         6039                end;
;         6040
;         6041            RAND_INDEX = .RAND_INDEX + .OFFSET; !Bump the random index if random data
;         6042            end;
;         6043
;         6044        end;
;         6045
;         6046    end;


                                        .SBTTL  FB.DATA.CHIPS ROUTINE DECLARATIONS
021074  004167  000000G          FB.DATA.CHIPS:
                                        JSR     R1,$SAVE5                       ;                               5922
021100  162706  000034                  SUB     #34,SP
021104  005067  000000G                  CLR     SEED1                           ;                               5961
021110  012767  001233  000000G         MOV     #1233,SEED2                     ;                               5962
021116  012767  007622  000000G         MOV     #7622,SEED3                     ;                               5963
021124  005066  000006                  CLR     6(SP)                           ; OFFSET                        5964
021130  016705  040436'                 MOV     RAND.PASS,R5                    ;                               5966
021134  010566  000012                  MOV     R5,12(SP)
021140  005266  000012                  INC     12(SP)
021144  005001                          CLR     R1                              ; PAT.SEL
021146  000167  000732                  JMP     16$
021152  005066  000004          1$:     CLR     4(SP)                           ; RAND.INDEX                    5968
021156  005701                          TST     R1                              ; PAT.SEL                       5970
021160  001004                          BNE     2$
021162  016646  000052                  MOV     52(SP),-(SP)                    ; TEMP.ARR$BNK.SE,*             5974
021166  005046                          CLR     -(SP)
021170  000407                          BR      3$
021172  020127  000001          2$:     CMP     R1,#1                           ; PAT.SEL,*                     5970
021176  001010                          BNE     4$
021200  016646  000052                  MOV     52(SP),-(SP)                    ; TEMP.ARR$BNK.SE,*             5977
021204  012746  177777                  MOV     #-1,-(SP)
021210  004767  164214          3$:     JSR     PC,L.1.0.DATA
021214  005726                          TST     (SP)+
021216  000407                          BR      5$                              ;                               5970
021220  016646  000052          4$:     MOV     52(SP),-(SP)                    ; TEMP.ARR$BNK.SE,*             5981
021224  004767  164070                  JSR     PC,L.RANDOM.DATA
021230  012766  000002  000010          MOV     #2,10(SP)                       ; *,OFFSET                      5982
```

```
021236  016766  040416' 000016          5$:     MOV     BNK.NUM.SEC,16(SP)              ;                                       5986
021244  005066  000002                          CLR     2(SP)                           ; SEC.CNT
021250  000167  000610                          JMP     14$
021254  104422                          6$:     TRAP    22                              ;                                       5987
021256  012746  177400                          MOV     #-400,-(SP)                     ;                                       5989
021262  011667  040426'                        MOV     (SP),SIZE
021266  016603  000056                          MOV     56(SP),R3                       ; TEMP.ARR$BNK.SE,*
021272  066603  000004                          ADD     4(SP),R3                        ; SEC.CNT,*
021276  010367  040430'                        MOV     R3,DST
021302  010346                                  MOV     R3,-(SP)
021304  016604  000012                          MOV     12(SP),R4                       ; RAND.INDEX,*                          5990
021310  006304                                  ASL     R4
021312  062704  002234'                        ADD     #WRT.BUF,R4
021316  010467  040432'                        MOV     R4,SRC
021322  010446                                  MOV     R4,-(SP)                        ;                                       5989
021324  004767  161610                          JSR     PC,WRT.CHK.TRANSFE
021330  016705  040374'                        MOV     ML.ADDR,R5                      ;                                       5992
021334  016566  000042  000042                 MOV     42(R5),42(SP)                   ; *,ML.REG
021342  032766  020000  000042                 BIT     #20000,42(SP)                   ; *,ML.REG
021350  001427                                  BEQ     7$
021352  016705  040374'                        MOV     ML.ADDR,R5                      ;                                       5995
021356  016566  000042  000040                 MOV     42(R5),40(SP)                   ; *,ML.REG
021364  016605  000040                          MOV     40(SP),R5                       ; ML.REG,*
021370  006205                                  ASR     R5
021372  006205                                  ASR     R5
021374  006205                                  ASR     R5
021376  006205                                  ASR     R5
021400  006205                                  ASR     R5
021402  006205                                  ASR     R5
021404  042705  177700                          BIC     #177700,R5
021410  010566  000012                          MOV     R5,12(SP)                       ; *,FAIL.CHIP
021414  010546                                  MOV     R5,-(SP)                        ; FAIL.CHIP,*                           5996
021416  010146                                  MOV     R1,-(SP)                        ; PAT.SEL,*
021420  004767  164120                          JSR     PC,L.CHIP.TBL
021424  022626                                  CMP     (SP)+,(SP)+                     ;                                       5994
021426  000411                                  BR      8$                              ;                                       5992
021430  016705  040374'          7$:     MOV     ML.ADDR,R5                      ;                                       6001
021434  016566  000042  000036                 MOV     42(R5),36(SP)                   ; *,ML.REG
021442  100003                                  BPL     8$
021444  152767  000002  040422'                BISB    #2,FLG.REG                      ;                                       6004
021452  016705  040374'          8$:     MOV     ML.ADDR,R5                      ;                                       6009
021456  016566  000010  000034                 MOV     10(R5),34(SP)                   ; *,ML.REG
021464  032766  040000  000034                 BIT     #40000,34(SP)                   ; *,ML.REG
021472  001565                                  BEQ     13$
021474  016705  040374'                        MOV     ML.ADDR,R5                      ;                                       6011
021500  016566  000010  000032                 MOV     10(R5),32(SP)                   ; *,ML.REG
021506  016604  000032                          MOV     32(SP),R4                       ; ML.REG,MLREG
021512  152704  000040                          BISB    #40,R4                          ; *,MLREG
021516  016705  040374'                        MOV     ML.ADDR,R5
021522  010465  000010                          MOV     R4,10(R5)                       ; MLREG,*
021526  016705  040374'                        MOV     ML.ADDR,R5
021532  016566  000010  000030                 MOV     10(R5),30(SP)                   ; *,ML.REG
```

```
021540  016604  000030                  MOV     30(SP),R4                       ; ML.REG,MLREG
021544  016705  040400'                 MOV     ML.DUT,R5
021550  042705  177770                  BIC     #177770,R5
021554  142704  000007                  BICB    #7,R4                           ; *,MLREG
021560  050504                          BIS     R5,R4                           ; *,MLREG
021562  016705  040374'                 MOV     ML.ADDR,R5
021566  010465  000010                  MOV     R4,10(R5)                       ; MLREG,*
021572  005066  000020                  CLR     20(SP)                          ; BITS.TSTED                    6013
021576  012746  177400                  MOV     #-400,-(SP)                     ;                               6014
021602  011667  040426'                 MOV     (SP),SIZE
021606  012746  007230'                 MOV     #RD.BUF,-(SP)
021612  011667  040430'                 MOV     (SP),DST
021616  010367  040432'                 MOV     R3,SRC
021622  010346                          MOV     R3,-(SP)
021624  004767  162540                  JSR     PC,RD.TRANSFER
021630  005002                          CLR     R2                              ; BUF.INDEX                     6016
021632  010204                  9$:     MOV     R2,R4                           ; BUF.INDEX,*                   6019
021634  066604  000022                  ADD     22(SP),R4                       ; RAND.INDEX,*
021640  006304                          ASL     R4
021642  010205                          MOV     R2,R5                           ; BUF.INDEX,*
021644  006305                          ASL     R5
021646  026465  002234' 007230'         CMP     WRT.BUF(R4),RD.BUF(R5)
021654  001463                          BEQ     12$
021656  016403  002234'                 MOV     WRT.BUF(R4),R3                  ;                               6022
021662  016546  007230'                 MOV     RD.BUF(R5),-(SP)
021666  040316                          BIC     R3,(SP)
021670  046503  007230'                 BIC     RD.BUF(R5),R3
021674  052603                          BIS     (SP)+,R3
021676  010366  000034                  MOV     R3,34(SP)                       ; *,BAD.BITS
021702  005005                          CLR     R5                              ; BIT.INDEX                     6024
021704  010504                  10$:    MOV     R5,R4                           ; BIT.INDEX,*                   6026
021706  006204                          ASR     R4
021710  006204                          ASR     R4
021712  006204                          ASR     R4
021714  012703  000034                  MOV     #34,R3
021720  060603                          ADD     SP,R3                           ; BAD.BITS,*
021722  060304                          ADD     R3,R4
021724  010446                          MOV     R4,-(SP)
021726  010546                          MOV     R5,-(SP)                        ; BIT.INDEX,*
021730  042716  177770                  BIC     #177770,(SP)
021734  012746  000001                  MOV     #1,-(SP)
021740  005046                          CLR     -(SP)
021742  004767  000000G                 JSR     PC,BL$GT2
021746  062706  000010                  ADD     #10,SP
021752  005300                          DEC     R0
021754  001017                          BNE     11$
021756  010546                          MOV     R5,-(SP)                        ; BIT.INDEX,*                   6029
021760  066616  000030                  ADD     30(SP),(SP)                     ; BITS.TSTED,*
021764  012746  000044                  MOV     #44,-(SP)
021770  004767  000000G                 JSR     PC,BL$MOD
021774  010066  000024                  MOV     R0,24(SP)                       ; *,FAIL.CHIP
022000  010016                          MOV     R0,(SP)                         ; FAIL.CHIP,*                   6031
```

```
022002  010146                          MOV     R1,-(SP)                        ; PAT.SEL,*
022004  004767  163534                  JSR     PC,L.CHIP.TBL
022010  062706  000006                  ADD     #6,SP                           ;                       6028
022014  005205                  11$:    INC     R5                              ; BIT.INDEX             6024
022016  020527  000017                  CMP     R5,#17                          ; BIT.INDEX,*
022022  003730                          BLE     10$
022024  062766  000020  000026  12$:    ADD     #20,26(SP)                      ; *,BITS.TSTED          6036
022032  005202                          INC     R2                              ; BUF.INDEX             6016
022034  020227  000377                  CMP     R2,#377                         ; BUF.INDEX,*
022040  003674                          BLE     9$
022042  062706  000006                  ADD     #6,SP                           ;                       6011
022046  066666  000016  000014  13$:    ADD     16(SP),14(SP)                   ; OFFSET,RAND.INDEX     6041
022054  062706  000006                  ADD     #6,SP                           ;                       5987
022060  005266  000002                  INC     2(SP)                           ; SEC.CNT               5986
022064  026666  000002  000016  14$:    CMP     2(SP),16(SP)                    ; SEC.CNT,*
022072  003002                          BGT     15$
022074  000167  177154                  JMP     6$
022100  005726                  15$:    TST     (SP)+                           ;                       5967
022102  005201                          INC     R1                              ; PAT.SEL               5966
022104  020166  000012          16$:    CMP     R1,12(SP)                       ; PAT.SEL,*
022110  003002                          BGT     17$
022112  000167  177034                  JMP     1$
022116  062706  000034          17$:    ADD     #34,SP                          ;                       5922
022122  000207                          RTS     PC

; Routine Size:  268 words
; Maximum stack depth per invocation:  31 words
```

```
;         6047  routine FB_CRC_CHIPS (TEMP_ARR$BNK_SEL) : novalue =
;         6048      begin
;         6049
;         6050  !++
;         6051  ! Functional Description:
;         6052  !       This global routine finds all the additional failing
;         6053  !       crc chips for a selected bank and is only called if
;         6054  !       a unc error occured during the find bad data chip
;         6055  !       global routine.
;         6056  !
;         6057  !       If a bad crc chip is found then it is stored
;         6058  !       into the bad chip table along with its failing
;         6059  !       data pattern.
;         6060  !
;         6061  ! Formal Parameters:
;         6062  !       ARR$BNK_SEL:
;         6063  !               Stores array and bank select address.
;         6064  !
;         6065  ! Implicit Inputs:
;         6066  !       RET_STATUS
;         6067  !
;         6068  ! Implicit Outputs:
;         6069  !       none
;         6070  !
;         6071  ! Completion codes:
;         6072  !       none
;         6073  !
;         6074  ! Side Effects:
;         6075  !       none
;         6076  !--
;         6077
;         6078      local
;         6079          RAND_INDEX,                         !Starts sector transfers at random write buffer words
;         6080          OFFSET,                             !Random data offset
;         6081          PD_SAVE : bitvector [16];           !Save location for nibble prom data
;         6082
;         6083      OFFSET = ZERO;                          !Clear the random data offset for 1's and 0's data
;         6084      SEED1 = ZERO;                           !Init the random data seed1
;         6085      SEED2 = %o'1233';                       !Init the random data seed2
;         6086      SEED3 = %o'7622';                       !Init the random data seed3
;         6087
;         6088      incr CRC_SEL from CRC_P to CRC_B do     !Find bad crc bits p, a, b
;         6089          begin
;         6090
;         6091          if not .CHIP_TBL [.CRC_SEL, FAULT]  !Did this crc bit fail during 'fb_data_chips'
;         6092          then
;         6093              begin
;         6094
;         6095              incr PAT_SEL from 0 to ONE + .RAND_PASS do  !Loop thru all patterns
;         6096                  begin
;         6097                  RAND_INDEX = ZERO;          !Reset the random index
;         6098
```

```
;       6099                    selectone .PAT_SEL of           !Select a pattern
;       6100                        set
;       6101
;       6102                        [0] :
;       6103                            DM_1_0_LOAD (.TEMP_ARR$BNK_SEL, ZEROES);        !Load zeroes data
;       6104
;       6105                        [1] :
;       6106                            DM_1_0_LOAD (.TEMP_ARR$BNK_SEL, ONES);  !Load ones data
;       6107
;       6108                        [otherwise] :
;       6109                            begin
;       6110                            DM_RAND_LOAD (.TEMP_ARR$BNK_SEL);       !Load random data
;       6111                            OFFSET = 2;             !Enable the random data offset
;       6112                            end;
;       6113                        tes;
;       6114
;       6115                    DM_RD_TRANSFER (SIZE = -256, DST = RD_BUF, SRC = .TEMP_ARR$BNK_SEL);
;       6116
;       6117                    case .CRC_SEL from CRC_P to CRC_B of    !Select the failing crc bit
;       6118                        set
;       6119
;       6120                        [36] :                      !Test crc bit p
;       6121                            begin
;       6122
;       6123                            incr SEC_CNT from 0 to .BNK_NUM_SEC do  !Read thru all sectors
;       6124                                begin
;       6125                                BREAK;
;       6126
;       6127                                incr WRD_CNT from 0 to 127 do       !Read thru all words in sector
;       6128                                    begin
;       6129                                    PD_SAVE = .ML_ADDR [MLPD, PD_REG];
;       6130                                                !Read out the prom data for this word
;       6131                                    DAT_CLK;        !Clock the data into the diag reg's
;       6132
;       6133                                    if not .PD_SAVE [CRC_NIBBLE]    !Is this a good nibble
;       6134                                    then
;       6135                                        begin
;       6136
;       6137                                        if .ML_ADDR [MLE2, B_36] neq .WRT_BUF [(.RAND_INDEX + .WRD_CNT), BIT_12]
;       6138                                        then
;       6139                                            begin
;       6140                                            L_CHIP_TBL (.CRC_SEL, .PAT_SEL);
;       6141                                                        !Save the failing chip and pattern
;       6142                                            end;
;       6143
;       6144                                        end;
;       6145
;       6146                                    end;
;       6147
;       6148                                RAND_INDEX = .RAND_INDEX + .OFFSET; !Bump the random index
;       6149                                end;
;       6150
```

```
;          6151                        end;
;          6152
;          6153                    [37] :                          !Test crc bit a
;          6154                        begin
;          6155
;          6156                        incr SEC_CNT from 0 to .BNK_NUM_SEC do  !Read thru all sectors
;          6157                            begin
;          6158                            BREAK;
;          6159
;          6160                            incr WRD_CNT from 0 to 127 do       !Read thru all words in this sector
;          6161                                begin
;          6162                                PD_SAVE = .ML_ADDR [MLPD, PD_REG];      !Get this words prom data
;          6163                                DAT_CLK;        !Clock out data into the diag reg's
;          6164
;          6165                                if not .PD_SAVE [CRC_NIBBLE]    !Is this a good nibble
;          6166                                then
;          6167                                    begin
;          6168
;          6169                                    if .ML_ADDR [MLE2, B_37] neq .WRT_BUF [(.RAND_INDEX + .WRD_CNT), BIT_13]
;          6170                                    then
;          6171                                        begin
;          6172                                        L_CHIP_TBL (.CRC_SEL, .PAT_SEL);
;          6173                                                !Save the bad chip and pattern
;          6174                                        end;
;          6175
;          6176                                    end;
;          6177
;          6178                                end;
;          6179
;          6180                            RAND_INDEX = .RAND_INDEX + .OFFSET; !Bump the random index
;          6181                            end;
;          6182
;          6183                        end;
;          6184
;          6185                    [38] :                          !Test crc bit b
;          6186                        begin
;          6187
;          6188                        incr SEC_CNT from 0 to .BNK_NUM_SEC do  !Read all sectors
;          6189                            begin
;          6190                            BREAK;
;          6191
;          6192                            incr WRD_CNT from 0 to 127 do       !Read all words in this sector
;          6193                                begin
;          6194                                PD_SAVE = .ML_ADDR [MLPD, PD_REG];      !Get the prom data for
;          6195                                DAT_CLK;        !Clock out data into diag reg's
;          6196
;          6197                                if not .PD_SAVE [CRC_NIBBLE]    !Is this a good nibble
;          6198                                then
;          6199                                    begin
;          6200
;          6201                                    if .ML_ADDR [MLE2, B_38] neq .WRT_BUF [(.RAND_INDEX + .WRD_CNT), BIT_14]
;          6202                                    then
```

```
;       6203                                        begin
;       6204                                        L_CHIP_TBL (.CRC_SEL, .PAT_SEL);
;       6205                                                !Save the bad chip and pattern
;       6206                                        end;
;       6207
;       6208                                    end;
;       6209
;       6210                                end;
;       6211
;       6212                            RAND_INDEX = .RAND_INDEX + .OFFSET; !Bump the random index
;       6213                            end;
;       6214
;       6215                        end
;       6216                    tes;
;       6217
;       6218                end;
;       6219
;       6220            end;
;       6221
;       6222        end;
;       6223
;       6224    CLR_MBUS;                               !Clear the single step dma mode
;       6225    end;


                                    .SBTTL  FB.CRC.CHIPS ROUTINE DECLARATIONS
022124  004167  000000G        FB.CRC.CHIPS:
                                    JSR     R1,$SAVE5                       ;                       6047
022130  162706  000042                SUB     #42,SP
022134  005066  000002                CLR     2(SP)                           ; OFFSET                6083
022140  005067  000000G               CLR     SEED1                           ;                       6084
022144  012767  001233  000000G       MOV     #1233,SEED2                     ;                       6085
022152  012767  007622  000000G       MOV     #7622,SEED3                     ;                       6086
022160  012701  000044                MOV     #44,R1                          ; *,CRC.SEL             6088
022164  010105                1$:     MOV     R1,R5                           ; CRC.SEL,*             6091
022166  006305                        ASL     R5
022170  005765  030230'               TST     CHIP.TBL(R5)
022174  100002                        BPL     2$
022176  000167  001052                JMP     27$
022202  016705  040436'        2$:     MOV     RAND.PASS,R5                    ;                       6095
022206  010566  000010                MOV     R5,10(SP)
022212  005266  000010                INC     10(SP)
022216  005002                        CLR     R2                              ; PAT.SEL
022220  000167  001016                JMP     26$
022224  005016                3$:     CLR     (SP)                            ; RAND.INDEX            6097
022226  005702                        TST     R2                              ; PAT.SEL               6099
022230  001004                        BNE     4$
022232  016646  000060                MOV     60(SP),-(SP)                    ; TEMP.ARRSBNK.SE,*     6103
022236  005046                        CLR     -(SP)
022240  000407                        BR      5$
022242  020227  000001        4$:     CMP     R2,#1                           ; PAT.SEL,*             6099
022246  001010                        BNE     6$
```

```
022250  016646  000060                  MOV     60(SP),-(SP)            ; TEMP.ARR$BNK.SE,*             6106
022254  012746  177777                  MOV     #-1,-(SP)
022260  004767  160304          5$:     JSR     PC,DM.1.0.LOAD
022264  005726                          TST     (SP)+
022266  000407                          BR      7$                      ;                               6099
022270  016646  000060          6$:     MOV     60(SP),-(SP)            ; TEMP.ARR$BNK.SE,*             6110
022274  004767  157674                  JSR     PC,DM.RAND.LOAD
022300  012766  000002  000004          MOV     #2,4(SP)                ; *,OFFSET                      6111
022306  012716  177400          7$:     MOV     #-400,(SP)              ;                               6115
022312  011667  040426'                 MOV     (SP),SIZE
022316  012746  007230'                 MOV     #RD.BUF,-(SP)
022322  011667  040430'                 MOV     (SP),DST
022326  016646  000064                  MOV     64(SP),-(SP)            ; TEMP.ARR$BNK.SE,*
022332  011667  040432'                 MOV     (SP),SRC
022336  004767  157022                  JSR     PC,DM.RD.TRANSFER
022342  010105                          MOV     R1,R5                   ; CRC.SEL,*                     6117
022344  162705  000044                  SUB     #44,R5
022350  006305                          ASL     R5
022352  066507  022356'                 ADD     8$(R5),PC
022356  000006          8$:     .WORD   9$-8$
022360  000224                  .WORD   14$-8$
022362  000442                  .WORD   20$-8$
022364  016766  040416' 000014  9$:     MOV     BNK.NUM.SEC,14(SP)      ;                               6123
022372  005066  000012                  CLR     12(SP)                  ; SEC.CNT
022376  000474                          BR      13$
022400  104422          10$:    TRAP    22                              ;                               6124
022402  005003                          CLR     R3                      ; WRD.CNT                       6127
022404  016705  040374'         11$:    MOV     ML.ADDR,R5              ;                               6129
022410  016566  000046  000046          MOV     46(R5),46(SP)           ; *,ML.REG
022416  016666  000046  000042          MOV     46(SP),42(SP)           ; ML.REG,PD.SAVE
022424  016705  040374'                 MOV     ML.ADDR,R5
022430  016566  000024  000044          MOV     24(R5),44(SP)           ; *,ML.REG
022436  016604  000044                  MOV     44(SP),R4               ; ML.REG,MLREG
022442  152704  000020                  BISB    #20,R4                  ; *,MLREG
022446  016705  040374'                 MOV     ML.ADDR,R5
022452  010465  000024                  MOV     R4,24(R5)               ; MLREG,*
022456  132766  000002  000043          BITB    #2,43(SP)               ; *,PD.SAVE+1                   6133
022464  001030                          BNE     12$
022466  016705  040374'                 MOV     ML.ADDR,R5              ;                               6137
022472  016566  000034  000040          MOV     34(R5),40(SP)           ; *,ML.REG
022500  010305                          MOV     R3,R5                   ; WRD.CNT,*
022502  066605  000006                  ADD     6(SP),R5                ; RAND.INDEX,*
022506  006305                          ASL     R5
022510  016500  002234'                 MOV     WRT.BUF(R5),R0
022514  042700  167777                  BIC     #167777,R0
022520  016604  000040                  MOV     40(SP),R4               ; ML.REG,*
022524  042704  167777                  BIC     #167777,R4
022530  020400                          CMP     R4,R0
022532  001405                          BEQ     12$
022534  010146                          MOV     R1,-(SP)                ; CRC.SEL,*                     6140
022536  010246                          MOV     R2,-(SP)                ; PAT.SEL,*
022540  004767  163000                  JSR     PC,L.CHIP.TBL
```

```
022544  022626                          CMP     (SP)+,(SP)+                     ;                               6139
022546  005203                  12$:    INC     R3                              ; WRD.CNT                       6127
022550  020327  000177                  CMP     R3,#177                         ; WRD.CNT,*
022554  003713                          BLE     11$
022556  066666  000010  000006          ADD     10(SP),6(SP)                    ; OFFSET,RAND.INDEX             6148
022564  005266  000012                  INC     12(SP)                          ; SEC.CNT                       6123
022570  026666  000012  000014  13$:    CMP     12(SP),14(SP)                   ; SEC.CNT,*
022576  003700                          BLE     10$
022600  000506                          BR      19$                             ;                               6117
022602  016766  040416' 000014  14$:    MOV     BNK.NUM.SEC,14(SP)              ;                               6156
022610  005066  000012                  CLR     12(SP)                          ; SEC.CNT
022614  000474                          BR      18$
022616  104422                  15$:    TRAP    22                              ;                               6157
022620  005003                          CLR     R3                              ; WRD.CNT                       6160
022622  016705  040374'         16$:    MOV     ML.ADDR,R5                      ;                               6162
022626  016566  000046  000036          MOV     46(R5),36(SP)                   ; *,ML.REG
022634  016666  000036  000042          MOV     36(SP),42(SP)                   ; ML.REG,PD.SAVE
022642  016705  040374'                  MOV     ML.ADDR,R5
022646  016566  000024  000034          MOV     24(R5),34(SP)                   ; *,ML.REG
022654  016604  000034                  MOV     34(SP),R4                       ; ML.REG,MLREG
022660  152704  000020                  BISB    #20,R4                          ; *,MLREG
022664  016705  040374'                  MOV     ML.ADDR,R5
022670  010465  000024                  MOV     R4,24(R5)                       ; MLREG,*
022674  132766  000002  000043          BITB    #2,43(SP)                       ; *,PD.SAVE+1                   6165
022702  001030                          BNE     17$
022704  016705  040374'                  MOV     ML.ADDR,R5                      ;                               6169
022710  016566  000034  000032          MOV     34(R5),32(SP)                   ; *,ML.REG
022716  010305                          MOV     R3,R5                           ; WRD.CNT,*
022720  066605  000006                  ADD     6(SP),R5                        ; RAND.INDEX,*
022724  006305                          ASL     R5
022726  016500  002234'                  MOV     WRT.BUF(R5),R0
022732  042700  157777                  BIC     #157777,R0
022736  016604  000032                  MOV     32(SP),R4                       ; ML.REG,*
022742  042704  157777                  BIC     #157777,R4
022746  020400                          CMP     R4,R0
022750  001405                          BEQ     17$
022752  010146                          MOV     R1,-(SP)                        ; CRC.SEL,*                     6172
022754  010246                          MOV     R2,-(SP)                        ; PAT.SEL,*
022756  004767  162562                  JSR     PC,L.CHIP.TBL
022762  022626                          CMP     (SP)+,(SP)+                     ;                               6171
022764  005203                  17$:    INC     R3                              ; WRD.CNT                       6160
022766  020327  000177                  CMP     R3,#177                         ; WRD.CNT,*
022772  003713                          BLE     16$
022774  066666  000010  000006          ADD     10(SP),6(SP)                    ; OFFSET,RAND.INDEX             6180
023002  005266  000012                  INC     12(SP)                          ; SEC.CNT                       6156
023006  026666  000012  000014  18$:    CMP     12(SP),14(SP)                   ; SEC.CNT,*
023014  003700                          BLE     15$
023016  000506                  19$:    BR      25$                             ;                               6117
023020  016766  040416' 000014  20$:    MOV     BNK.NUM.SEC,14(SP)              ;                               6188
023026  005066  000012                  CLR     12(SP)                          ; SEC.CNT
023032  000474                          BR      24$
023034  104422                  21$:    TRAP    22                              ;                               6189
```

```
023036  005003                          CLR     R3                              ; WRD.CNT                       6192
023040  016705  040374'                22$:    MOV     ML.ADDR,R5              ;                               6194
023044  016566  000046  000030                MOV     46(R5),30(SP)           ; *,ML.REG
023052  016666  000030  000042                MOV     30(SP),42(SP)           ; ML.REG,PD.SAVE
023060  016705  040374'                        MOV     ML.ADDR,R5
023064  016566  000024  000026                MOV     24(R5),26(SP)           ; *,ML.REG
023072  016604  000026                        MOV     26(SP),R4               ; ML.REG,MLREG
023076  152704  000020                        BISB    #20,R4                  ; *,MLREG
023102  016705  040374'                        MOV     ML.ADDR,R5
023106  010465  000024                        MOV     R4,24(R5)               ; MLREG,*
023112  132766  000002  000043                BITB    #2,43(SP)               ; *,PD.SAVE+1                   6197
023120  001030                              BNE     23$
023122  016705  040374'                        MOV     ML.ADDR,R5              ;                               6201
023126  016566  000034  000024                MOV     34(R5),24(SP)           ; *,ML.REG
023134  010305                              MOV     R3,R5                   ; WRD.CNT,*
023136  066605  000006                        ADD     6(SP),R5                ; RAND.INDEX,*
023142  006305                              ASL     R5
023144  016500  002234'                      MOV     WRT.BUF(R5),R0
023150  042700  137777                      BIC     #137777,R0
023154  016604  000024                      MOV     24(SP),R4               ; ML.REG,*
023160  042704  137777                      BIC     #137777,R4
023164  020400                              CMP     R4,R0
023166  001405                              BEQ     23$
023170  010146                              MOV     R1,-(SP)                ; CRC.SEL,*                     6204
023172  010246                              MOV     R2,-(SP)                ; PAT.SEL,*
023174  004767  162344                      JSR     PC,L.CHIP.TBL
023200  022626                              CMP     (SP)+,(SP)+             ;                               6203
023202  005203                  23$:        INC     R3                      ; WRD.CNT                       6192
023204  020327  000177                      CMP     R3,#177                 ; WRD.CNT,*
023210  003713                              BLE     22$
023212  066666  000010  000006              ADD     10(SP),6(SP)            ; OFFSET,RAND.INDEX             6212
023220  005266  000012                      INC     12(SP)                  ; SEC.CNT                       6188
023224  026666  000012  000014      24$:    CMP     12(SP),14(SP)           ; SEC.CNT,*
023232  003700                              BLE     21$
023234  062706  000006              25$:    ADD     #6,SP                   ;                               6096
023240  005202                              INC     R2                      ; PAT.SEL                       6095
023242  020266  000010              26$:    CMP     R2,10(SP)               ; PAT.SEL,*
023246  003002                              BGT     27$
023250  000167  176750                      JMP     3$
023254  005201                      27$:    INC     R1                      ; CRC.SEL                       6088
023256  020127  000046                      CMP     R1,#46                  ; CRC.SEL,*
023262  003002                              BGT     28$
023264  000167  176674                      JMP     1$
023270  016705  040374'              28$:    MOV     ML.ADDR,R5              ;                               6222
023274  016566  000010  000014              MOV     10(R5),14(SP)           ; *,ML.REG
023302  016604  000014                      MOV     14(SP),R4               ; ML.REG,MLREG
023306  152704  000040                      BISB    #40,R4                  ; *,MLREG
023312  016705  040374'                      MOV     ML.ADDR,R5
023316  010465  000010                      MOV     R4,10(R5)               ; MLREG,*
023322  016705  040374'                      MOV     ML.ADDR,R5
023326  016566  000010  000012              MOV     10(R5),12(SP)           ; *,ML.REG
023334  016604  000012                      MOV     12(SP),R4               ; ML.REG,MLREG
```

```
023340  016705  040400'          MOV     ML.DUT,R5
023344  042705  177770           BIC     #177770,R5
023350  142704  000007           BICB    #7,R4                   ; *,MLREG
023354  050504                   BIS     R5,R4                   ; *,MLREG
023356  016705  040374'          MOV     ML.ADDR,R5
023362  010465  000010           MOV     R4,10(R5)               ; MLREG,*
023366  062706  000042           ADD     #42,SP                  ;                       6047
023372  000207                   RTS     PC

; Routine Size:  340 words
; Maximum stack depth per invocation:  28 words
```

```
;          6226  routine FB_CHIPS (TEMP_ARR$BNK_SEL) : novalue =
;          6227      begin
;          6228
;          6229  !++
;          6230  ! Functional Description:
;          6231  !       This global routine calls the 'find bad data chip' and
;          6232  !       'find bad crc chip' global routines which will search
;          6233  !       this tested bank and find and log all failing chips.
;          6234  !
;          6235  !       If the fb_data_chip global routine detectes a unc
;          6236  !       error then the fb_crc_chip global routine is not
;          6237  !       called.
;          6238  !
;          6239  ! Formal Parameters:
;          6240  !       ARR$BNK_SEL:
;          6241  !               Stores the array and bank select address.
;          6242  !
;          6243  ! Implicit Inputs:
;          6244  !       FLG_REG
;          6245  !
;          6246  ! Implicit Outputs:
;          6247  !       none
;          6248  !
;          6249  ! Completion codes:
;          6250  !       none
;          6251  !
;          6252  ! Side Effects:
;          6253  !       none
;          6254  !--
;          6255
;          6256      FLG_REG [F_UNC_ERR_FLG] = CLR_FLG;          !Clear the uncorrectable error flag
;          6257      I_CHIP_TBL ();                              !Init the bad chip table
;          6258      FB_DATA_CHIPS (.TEMP_ARR$BNK_SEL);          !Find all the bad data chips 0-35 in this bank
;          6259
;          6260      if .FLG_REG [F_UNC_ERR_FLG] then FB_CRC_CHIPS (.TEMP_ARR$BNK_SEL);
;          6261
;          6262                                                  !Find all the bad crc chips in this bank
;          6263      end;


                                        .SBTTL  FB.CHIPS ROUTINE DECLARATIONS
023374  142767  000002  040422'  FB.CHIPS:
                                        BICB    #2,FLG.REG                      ;                       6256
023402  004767  161472                  JSR     PC,I.CHIP.TBL                   ;                       6257
023406  016646  000002                  MOV     2(SP),-(SP)                     ; TEMP.ARR$BNK.SE,*     6258
023412  004767  175456                  JSR     PC,FB.DATA.CHIPS
023416  032767  000002  040422'        BIT     #2,FLG.REG                      ;                       6260
023424  001405                          BEQ     1$
023426  016646  000004                  MOV     4(SP),-(SP)                     ; TEMP.ARR$BNK.SE,*
023432  004767  176466                  JSR     PC,FB.CRC.CHIPS
023436  005726                          TST     (SP)+
023440  005726                  1$:     TST     (SP)+                           ;                       6227
```

```
023442  000207                          RTS     PC                              ;                                       6226

; Routine Size:  20 words
; Maximum stack depth per invocation:  2 words
```

```
;        6264  routine PM_THIS_BANK (TEMP_ARRSBNK_SEL) : novalue =
;        6265      begin
;        6266
;        6267  !++
;        6268  ! Functional Description:
;        6269  !       This global routine loops on routines that will
;        6270  !       find newly failing chips, load the error map,
;        6271  !       interigate the error map and load the blast table
;        6272  !       with all the newly failing rows and columns
;        6273  !       addresses within this bank.
;        6274  !
;        6275  ! Formal Parameters:
;        6276  !       ARRSBNK_SEL:
;        6277  !               Stores the array and bank select address.
;        6278  !
;        6279  ! Implicit Inputs:
;        6280  !       CHIP_TBL, FLG_REG
;        6281  !
;        6282  ! Implicit Outputs:
;        6283  !       As each failing chip gets PM'ed the fault bit
;        6284  !       in the flag register gets cleared.
;        6285  !
;        6286  ! Completion codes:
;        6287  !       none
;        6288  !
;        6289  ! Side Effects:
;        6290  !       none
;        6291  !--
;        6292
;        6293      local
;        6294          BAD_CHIP,                           !Points to bad chips to be tested
;        6295          LST_PAT;                            !Last pattern test for this failing chip
;        6296
;        6297      BAD_CHIP = S_CHIP_TBL (ZERO);           !Find a bad chip to test
;        6298
;        6299      if .BAD_CHIP geq ZERO                   !Are there any chips to test
;        6300      then
;        6301          begin
;        6302          FLG_REG [F_ALL_BAD_CHIP] = CLR_FLG;     !Clear the all bad flag
;        6303
;        6304          do
;        6305              begin
;        6306              SEED1 = ZERO;                   !Init the random data seed1
;        6307              SEED2 = %o'1233';               !Init the random data seed2
;        6308              SEED3 = %o'7622';               !Init the ranomd data seed3
;        6309              FLG_REG [F_ERR_MAP_ENTERED] = CLR_FLG;      !Clear the entered error map flag
;        6310              BREAK;
;        6311              I_ERROR_MAP ();                 !Init the error map
;        6312              LST_PAT = ZERO;                 !Last pattern starts at zero
;        6313
;        6314              do                              !Test all failing patterns for this bad chip
;        6315                  begin
```

```
;          6316                    LST_PAT = L_FAILING_CHIP (.TEMP_ARR$BNK_SEL, .BAD_CHIP, .LST_PAT);
;          6317                    L_ERROR_MAP (.TEMP_ARR$BNK_SEL, .BAD_CHIP);
;          6318                                                    !Load the error map with failing rows and columns
;          6319                    end
;          6320                until .CHIP_TBL [.BAD_CHIP, PATS] eql ZEROES;       !Repeat until all patterns are tested
;          6321
;          6322                if .FLG_REG [F_ERR_MAP_ENTERED]     !Were any failing rows/cols detected
;          6323                then
;          6324                    begin
;          6325                    IN_ERROR_MAP (.TEMP_ARR$BNK_SEL, .BAD_CHIP);    !Interigate the error map
;          6326                    end
;          6327                else
;          6328                    begin
;          6329                    ERRSF (ERR_4, UNC_CHIP_MSG, 0); !Report the unconfirmed error
;        P 6330                    PRINTB (A_B_C_PRINT, .TEMP_ARR$BNK_SEL<.ARR_SEL_POS, ARR$SEL_SIZE>,
;          6331                        .TEMP_ARR$BNK_SEL<.BNK_SEL_POS, BNK$SEL_SIZE>, .BAD_CHIP);  !Report where
;          6332                    end;
;          6333
;          6334                CHIP_TBL [.BAD_CHIP, FAULT] = CLR_FLG;      !Clear this chips fault flag
;          6335                BAD_CHIP = S_CHIP_TBL (.BAD_CHIP);  !Find the next failing chip
;          6336                end
;          6337            until (.BAD_CHIP lss ZERO) or (.FLG_REG [F_ABORT_ARRAY]);       !Repeat until all bad chips tested
;          6338
;          6339            end;
;          6340
;          6341        end;


                                        .SBTTL  PM.THIS.BANK ROUTINE DECLARATIONS
023444  004167  000000G         PM.THIS.BANK:
                                        JSR     R1,$SAVE4                       ;                               6264
023450  005046                          CLR     -(SP)                           ;                               6297
023452  004767  170632                  JSR     PC,S.CHIP.TBL
023456  010003                          MOV     R0,R3                           ; *,BAD.CHIP
023460  002544                          BLT     5$                              ;                               6299
023462  142767  000020  040422'        BICB    #20,FLG.REG                     ;                               6302
023470  005067  000000G         1$:     CLR     SEED1                           ;                               6306
023474  012767  001233  000000G       MOV     #1233,SEED2                     ;                               6307
023502  012767  007622  000000G       MOV     #7622,SEED3                     ;                               6308
023510  142767  000004  040422'        BICB    #4,FLG.REG                      ;                               6309
023516  104422                          TRAP    22
023520  004767  161530                  JSR     PC,I.ERROR.MAP                  ;                               6311
023524  005004                          CLR     R4                              ; LST.PAT                       6312
023526  010301                          MOV     R3,R1                           ; BAD.CHIP,*                    6320
023530  006301                          ASL     R1
023532  016602  000016                  MOV     16(SP),R2                       ; TEMP.ARR$BNK.SE,*             6316
023536  010246                  2$:     MOV     R2,-(SP)
023540  010346                          MOV     R3,-(SP)                        ; BAD.CHIP,*
023542  010446                          MOV     R4,-(SP)                        ; LST.PAT,*
023544  004767  162220                  JSR     PC,L.FAILING.CHIP
023550  010004                          MOV     R0,R4                           ; *,LST.PAT
023552  010216                          MOV     R2,(SP)                         ;                               6317
```

```
023554  010346                          MOV     R3,-(SP)                ; BAD.CHIP,*
023556  004767  163042                  JSR     PC,L.ERROR.MAP
023562  062706  000010                  ADD     #10,SP                  ;                       6315
023566  032761  077777  030230'         BIT     #77777,CHIP.TBL(R1)     ;                       6320
023574  001360                          BNE     2$
023576  032767  000004  040422'         BIT     #4,FLG.REG              ;                       6322
023604  001405                          BEQ     3$
023606  010246                          MOV     R2,-(SP)                ;                       6325
023610  010346                          MOV     R3,-(SP)                ; BAD.CHIP,*
023612  004767  174546                  JSR     PC,IN.ERROR.MAP
023616  000447                          BR      4$                      ;                       6322
023620  104454                  3$:     TRAP    54                      ;                       6329
023622  000004                          .WORD   4
023624  001454'                         .WORD   UNC.CHIP.MSG
023626  000000                          .WORD   0
023630  010346                          MOV     R3,-(SP)                ; BAD.CHIP,*            6331
023632  012746  000022                  MOV     #22,-(SP)
023636  060616                          ADD     SP,(SP)                 ; TEMP.ARR$BNK.SE,*
023640  016746  040414'                 MOV     BNK.SEL.POS,-(SP)
023644  012746  000002                  MOV     #2,-(SP)
023650  005046                          CLR     -(SP)
023652  004767  000000G                 JSR     PC,BL$GT2
023656  022626                          CMP     (SP)+,(SP)+
023660  010066  000002                  MOV     R0,2(SP)
023664  012716  000024                  MOV     #24,(SP)
023670  060616                          ADD     SP,(SP)                 ; TEMP.ARR$BNK.SE,*
023672  016746  040412'                 MOV     ARR.SEL.POS,-(SP)
023676  012746  000004                  MOV     #4,-(SP)
023702  005046                          CLR     -(SP)
023704  004767  000000G                 JSR     PC,BL$GT2
023710  022626                          CMP     (SP)+,(SP)+
023712  010066  000002                  MOV     R0,2(SP)
023716  012716  003052'                 MOV     #A.B.C.PRINT,(SP)
023722  012746  000004                  MOV     #4,-(SP)
023726  010600                          MOV     SP,R0                   ; SP,*
023730  104414                          TRAP    14
023732  062706  000006                  ADD     #6,SP                   ;                       6328
023736  042761  100000  030230'  4$:    BIC     #100000,CHIP.TBL(R1)    ;                       6334
023744  010316                          MOV     R3,(SP)                 ; BAD.CHIP,*            6335
023746  004767  170336                  JSR     PC,S.CHIP.TBL
023752  010003                          MOV     R0,R3                   ; *,BAD.CHIP
023754  022626                          CMP     (SP)+,(SP)+             ;                       6305
023756  005703                          TST     R3                      ; BAD.CHIP              6337
023760  002404                          BLT     5$
023762  032767  000100  040422'         BIT     #100,FLG.REG
023770  001637                          BEQ     1$
023772  005726                  5$:     TST     (SP)+                   ;                       6265
023774  000207                          RTS     PC                      ;                       6264

; Routine Size:  109 words
; Maximum stack depth per invocation:  12 words
```

```
;       6342  routine CAL_CHK_SUM : novalue =
;       6343      begin
;       6344
;       6345  !++
;       6346  ! Functional Description:
;       6347  !       This global routine will calculate the new
;       6348  !       check sums values for the prom data to
;       6349  !       be blasted into the selected arrays
;       6350  !       proms.
;       6351  !
;       6352  ! Formal Parameters:
;       6353  !       none
;       6354  !
;       6355  ! Implicit Inputs:
;       6356  !       BAD_BNK_REG, BLAST_TBL, ERROR_MAP
;       6357  !
;       6358  ! Implicit Outputs:
;       6359  !       The error map and blast table is loaded with the new
;       6360  !       checksum values for the newly discovered failing row
;       6361  !       and column addresses.
;       6362  !
;       6363  ! Completion codes:
;       6364  !       none
;       6365  !
;       6366  ! Side Effects:
;       6367  !       none
;       6368  !--
;       6369
;       6370      local
;       6371          PD_SAVE : bitvector [16],               !Stores the prom data to be calculated
;       6372          PD_0_9_CNT,                             !Counts number of bits 0-9 set
;       6373          CHK_SUM_CNT,                            !Counts the old check sum value
;       6374          REMAINDER;                              !Stores the number of count to update the check sums with
;       6375
;       6376      label
;       6377          A;                                      !Leave loop label
;       6378
;       6379      map
;       6380          ERROR_MAP : blockvector [4, 512] volatile;      !Make the error map look like the blast table
;       6381
;       6382      incr BNK_SEL from 0 to 3 do                 !Look at all the banks
;       6383          begin
;       6384
;       6385          if .BAD_BNK_REG [.BNK_SEL]              !Is this a bad bank
;       6386          then
;       6387              begin
;       6388
;       6389              incr PD_WRD_SEL from 0 to .MAX_CHIP_COL*2 - 1 do    !Look at all the rows and columnsns
;       6390                  begin
;       6391
;       6392                  if .BLAST_TBL [.BNK_SEL, .PD_WRD_SEL, FULL_WRD] neq ZERO        !Is this  location enpty
;       6393                  then
```

```
;       6394  A :
;       6395                    begin
;       6396                    PD_0_9_CNT = ZERO;              !Clear the count
;       6397                    CHK_SUM_CNT = ZERO;             !Clear the count
;       6398                    PD_SAVE = .BLAST_TBL [.BNK_SEL, .PD_WRD_SEL, FULL_WRD];     !Get this prom dtat word
;       6399
;       6400                    incr CNT from 0 to 9 do     !Count the # of bits set in 0 - 9
;       6401
;       6402                        if .PD_SAVE [.CNT] then PD_0_9_CNT = .PD_0_9_CNT + 1;
;       6403
;       6404                    CHK_SUM_CNT = .PD_SAVE<10, 3>;      !Get the old count
;       6405                    CHK_SUM_CNT = .CHK_SUM_CNT + .PD_SAVE<13, 1>;
;       6406                    CHK_SUM_CNT = .CHK_SUM_CNT + .PD_SAVE<14, 1>;
;       6407                    CHK_SUM_CNT = .CHK_SUM_CNT + .PD_SAVE<15, 1>;
;       6408                    REMAINDER = .PD_0_9_CNT - .CHK_SUM_CNT;     !How many more to enter
;       6409
;       6410                    if .REMAINDER geq 4         !Do we add in more than 3
;       6411                    then
;       6412                        begin
;       6413
;       6414                        if .PD_SAVE<12, 1> eql ZERO     !Is this bit already set
;       6415                        then
;       6416                            begin
;       6417                            PD_SAVE<12, 1> = SET_FLG;   !Set this bit
;       6418                            REMAINDER = .REMAINDER - 4; !Subtract the weight
;       6419
;       6420                            if .REMAINDER eql ZERO      !Are we all done
;       6421                            then
;       6422                                begin
;       6423                                BLAST_TBL [.BNK_SEL, .PD_WRD_SEL, FULL_WRD] = .PD_SAVE;
;       6424                                ERROR_MAP [.BNK_SEL, .PD_WRD_SEL, FULL_WRD] = .BLAST_TBL [.BNK_SEL,
;       6425                                    .PD_WRD_SEL, FULL_WRD];
;       6426                                leave A;        !Leave the loop
;       6427                                end;
;       6428
;       6429                            end;
;       6430
;       6431                        end;
;       6432
;       6433                    if .REMAINDER geq 2         !Do we add in more than 1
;       6434                    then
;       6435                        begin
;       6436
;       6437                        if .PD_SAVE<11, 1> eql ZERO     !Is this bit already set
;       6438                        then
;       6439                            begin
;       6440                            PD_SAVE<11, 1> = SET_FLG;   !Set this bit
;       6441                            REMAINDER = .REMAINDER - 2; !Subtract the weight
;       6442
;       6443                            if .REMAINDER eql ZERO      !Are we done
;       6444                            then
;       6445                                begin
```

```
;          6446                                BLAST_TBL [.BNK_SEL, .PD_WRD_SEL, FULL_WRD] = .PD_SAVE;
;          6447                                ERROR_MAP [.BNK_SEL, .PD_WRD_SEL, FULL_WRD] = .BLAST_TBL [.BNK_SEL,
;          6448                                    .PD_WRD_SEL, FULL_WRD];
;          6449                                leave A;            !Leave the loop
;          6450                                end;
;          6451
;          6452                            end;
;          6453
;          6454                        end;
;          6455
;          6456                    if .REMAINDER geq 1             !Do we add more than 0
;          6457                    then
;          6458                        begin
;          6459
;          6460                        if .PD_SAVE<10, 1> eql ZERO     !Is this bit already set
;          6461                        then
;          6462                            begin
;          6463                            PD_SAVE<10, 1> = SET_FLG;   !Set this bit
;          6464                            REMAINDER = .REMAINDER - 1; !Subtract the weight
;          6465
;          6466                            if .REMAINDER eql ZERO      !Are we done
;          6467                            then
;          6468                                begin
;          6469                                BLAST_TBL [.BNK_SEL, .PD_WRD_SEL, FULL_WRD] = .PD_SAVE;
;          6470                                ERROR_MAP [.BNK_SEL, .PD_WRD_SEL, FULL_WRD] = .BLAST_TBL [.BNK_SEL,
;          6471                                    .PD_WRD_SEL, FULL_WRD];
;          6472                                leave A;            !Leave the loop
;          6473                                end;
;          6474
;          6475                            end;
;          6476
;          6477                        end;
;          6478
;          6479                    if .PD_SAVE<13, 1> eql ZERO !Is this bit already set
;          6480                    then
;          6481                        begin
;          6482                        PD_SAVE<13, 1> = SET_FLG;       !Set this bit
;          6483                        REMAINDER = .REMAINDER - 1;
;          6484
;          6485                        if .REMAINDER eql ZERO  !Are we done
;          6486                        then
;          6487                            begin
;          6488                            BLAST_TBL [.BNK_SEL, .PD_WRD_SEL, FULL_WRD] = .PD_SAVE;
;          6489                            ERROR_MAP [.BNK_SEL, .PD_WRD_SEL, FULL_WRD] = .BLAST_TBL [.BNK_SEL,
;          6490                                .PD_WRD_SEL, FULL_WRD];
;          6491                            leave A;                ![eave the loop
;          6492                            end;
;          6493
;          6494                        end;
;          6495
;          6496                    if .PD_SAVE<14, 1> eql ZERO !Is this bit already set
;          6497                    then
```

```
;       6498                        begin
;       6499                        PD_SAVE<14, 1> = SET_FLG;       !Set this bit
;       6500                        REMAINDER = .REMAINDER - 1;     !Subtract the weight
;       6501
;       6502                        if .REMAINDER eql ZERO  !Are we done
;       6503                        then
;       6504                            begin
;       6505                            BLAST_TBL [.BNK_SEL, .PD_WRD_SEL, FULL_WRD] = .PD_SAVE;
;       6506                            ERROR_MAP [.BNK_SEL, .PD_WRD_SEL, FULL_WRD] = .BLAST_TBL [.BNK_SEL,
;       6507                                .PD_WRD_SEL, FULL_WRD];
;       6508                            leave A;                !Leave the loop
;       6509                            end;
;       6510
;       6511                        end;
;       6512
;       6513                    PD_SAVE<15, 1> = SET_FLG;   !Set the bit
;       6514                    BLAST_TBL [.BNK_SEL, .PD_WRD_SEL, FULL_WRD] = .PD_SAVE;
;       6515                    ERROR_MAP [.BNK_SEL, .PD_WRD_SEL, FULL_WRD] = .BLAST_TBL [.BNK_SEL, .PD_WRD_SEL,
;       6516                        FULL_WRD];
;       6517                    end;
;       6518
;       6519                end;
;       6520
;       6521            end;
;       6522
;       6523        end;
;       6524
;       6525    end;


                                .SBTTL  CAL.CHK.SUM ROUTINE DECLARATIONS
023776  004167  000000G         CAL.CHK.SUM:
                                JSR     R1,$SAVE5                       ;                       6342
024002  162706  000016                  SUB     #16,SP
024006  005046                  CLR     -(SP)                           ; BNK.SEL               6382
024010  011600          1$:     MOV     (SP),R0                         ; BNK.SEL,*             6385
024012  006200                  ASR     R0
024014  006200                  ASR     R0
024016  006200                  ASR     R0
024020  062700  040420'          ADD     #BAD.BNK.REG,R0
024024  010046                  MOV     R0,-(SP)
024026  016646  000002          MOV     2(SP),-(SP)                     ; BNK.SEL,*
024032  042716  177770          BIC     #177770,(SP)
024036  012746  000001          MOV     #1,-(SP)
024042  005046                  CLR     -(SP)
024044  004767  000000G         JSR     PC,BL$GT2
024050  062706  000010          ADD     #10,SP
024054  006000                  ROR     R0
024056  103402                  BLO     2$
024060  000167  000530          JMP     17$
024064  016766  040376' 000014  2$:     MOV     MAX.CHIP.COL,14(SP)             ;               6389
024072  006366  000014          ASL     14(SP)
```

```
024076  011600                          MOV     (SP),R0                 ; BNK.SEL,*                     6392
024100  000300                          SWAB    R0
024102  105000                          CLRB    R0
024104  006300                          ASL     R0
024106  010066  000012                  MOV     R0,12(SP)
024112  005066  000004                  CLR     4(SP)                   ; PD.WRD.SEL                    6389
024116  000167  000456                  JMP     16$
024122  016605  000004          3$:     MOV     4(SP),R5                ; PD.WRD.SEL,*                  6392
024126  066605  000012                  ADD     12(SP),R5
024132  006305                          ASL     R5
024134  012766  030372' 000010          MOV     #BLAST.TBL,10(SP)
024142  060566  000010                  ADD     R5,10(SP)
024146  005776  000010                  TST     @10(SP)
024152  001002                          BNE     4$
024154  000167  000414                  JMP     15$
024160  005066  000006          4$:     CLR     6(SP)                   ; PD.0.9.CNT                    6396
024164  005066  000002                  CLR     2(SP)                   ; CHK.SUM.CNT                   6397
024170  017666  000010  000016          MOV     @10(SP),16(SP)          ; *,PD.SAVE                     6398
024176  005004                          CLR     R4                      ; CNT                           6400
024200  010403          5$:     MOV     R4,R3                   ; CNT,*                         6402
024202  006203                          ASR     R3
024204  006203                          ASR     R3
024206  006203                          ASR     R3
024210  012702  000016                  MOV     #16,R2
024214  060602                          ADD     SP,R2                   ; PD.SAVE,*
024216  060203                          ADD     R2,R3
024220  010346                          MOV     R3,-(SP)
024222  010446                          MOV     R4,-(SP)                ; CNT,*
024224  042716  177770                  BIC     #177770,(SP)
024230  012746  000001                  MOV     #1,-(SP)
024234  005046                          CLR     -(SP)
024236  004767  000000G                 JSR     PC,BL$GT2
024242  062706  000010                  ADD     #10,SP
024246  006000                          ROR     R0
024250  005566  000006                  ADC     6(SP)                   ; PD.0.9.CNT
024254  005204                          INC     R4                      ; CNT                           6400
024256  020427  000011                  CMP     R4,#11                  ; CNT,*
024262  003746                          BLE     5$
024264  016604  000016                  MOV     16(SP),R4               ; PD.SAVE,*                     6404
024270  006204                          ASR     R4
024272  006204                          ASR     R4
024274  000304                          SWAB    R4
024276  042704  177770                  BIC     #177770,R4
024302  010466  000002                  MOV     R4,2(SP)                ; *,CHK.SUM.CNT
024306  005004                          CLR     R4                      ;                               6405
024310  032766  020000  000016          BIT     #20000,16(SP)           ; *,PD.SAVE
024316  001401                          BEQ     6$
024320  005204                          INC     R4
024322  060466  000002          6$:     ADD     R4,2(SP)                ; *,CHK.SUM.CNT
024326  005004                          CLR     R4                      ;                               6406
024330  032766  040000  000016          BIT     #40000,16(SP)           ; *,PD.SAVE
024336  001401                          BEQ     7$
```

```
024340  005204                          INC     R4
024342  060466  000002          7$:     ADD     R4,2(SP)                ; *,CHK.SUM.CNT
024346  005004                          CLR     R4                      ;                       6407
024350  005766  000016                  TST     16(SP)                  ; PD.SAVE
024354  100001                          BPL     8$
024356  005204                          INC     R4
024360  060466  000002          8$:     ADD     R4,2(SP)                ; *,CHK.SUM.CNT
024364  016601  000006                  MOV     6(SP),R1                ; PD.0.9.CNT,REMAINDER  6408
024370  166601  000002                  SUB     2(SP),R1                ; CHK.SUM.CNT,REMAINDER
024374  020127  000004                  CMP     R1,#4                   ; REMAINDER,*           6410
024400  002412                          BLT     9$
024402  032766  010000  000016          BIT     #10000,16(SP)           ; *,PD.SAVE             6414
024410  001006                          BNE     9$
024412  052766  010000  000016          BIS     #10000,16(SP)           ; *,PD.SAVE             6417
024420  162701  000004                  SUB     #4,R1                   ; *,REMAINDER           6418
024424  001455                          BEQ     14$                     ;                       6423
024426  020127  000002          9$:     CMP     R1,#2                   ; REMAINDER,*           6433
024432  002412                          BLT     10$
024434  032766  004000  000016          BIT     #4000,16(SP)            ; *,PD.SAVE             6437
024442  001006                          BNE     10$
024444  052766  004000  000016          BIS     #4000,16(SP)            ; *,PD.SAVE             6440
024452  162701  000002                  SUB     #2,R1                   ; *,REMAINDER           6441
024456  001440                          BEQ     14$                     ;                       6446
024460  005701                  10$:    TST     R1                      ; REMAINDER             6456
024462  003411                          BLE     11$
024464  032766  002000  000016          BIT     #2000,16(SP)            ; *,PD.SAVE             6460
024472  001005                          BNE     11$
024474  052766  002000  000016          BIS     #2000,16(SP)            ; *,PD.SAVE             6463
024502  005301                          DEC     R1                      ; REMAINDER             6464
024504  001425                          BEQ     14$                     ;                       6469
024506  032766  020000  000016  11$:    BIT     #20000,16(SP)           ; *,PD.SAVE             6479
024514  001005                          BNE     12$
024516  052766  020000  000016          BIS     #20000,16(SP)           ; *,PD.SAVE             6482
024524  005301                          DEC     R1                      ; REMAINDER             6483
024526  001414                          BEQ     14$                     ;                       6488
024530  032766  040000  000016  12$:    BIT     #40000,16(SP)           ; *,PD.SAVE             6496
024536  001005                          BNE     13$
024540  052766  040000  000016          BIS     #40000,16(SP)           ; *,PD.SAVE             6499
024546  005301                          DEC     R1                      ; REMAINDER             6500
024550  001403                          BEQ     14$                     ;                       6505
024552  052766  100000  000016  13$:    BIS     #100000,16(SP)          ; *,PD.SAVE             6513
024560  016676  000016  000010  14$:    MOV     16(SP),@10(SP)          ; PD.SAVE,*             6514
024566  017665  000010  010230'        MOV     @10(SP),ERROR.MAP(R5)   ;                       6515
024574  005266  000004          15$:    INC     4(SP)                   ; PD.WRD.SEL            6389
024600  026666  000004  000014  16$:    CMP     4(SP),14(SP)            ; PD.WRD.SEL,*
024606  002002                          BGE     17$
024610  000167  177306                  JMP     3$
024614  005216                  17$:    INC     (SP)                    ; BNK.SEL               6382
024616  021627  000003                  CMP     (SP),#3                 ; BNK.SEL,*
024622  003002                          BGT     18$
024624  000167  177160                  JMP     1$
024630  062706  000020          18$:    ADD     #20,SP                  ;                       6342
```

```
024634  000207                                  RTS     PC

; Routine Size:  208 words
; Maximum stack depth per invocation:  18 words
```

```
;          routine BL_PROMS (ARR$BNK_SEL) : novalue =
;              begin
;
;          !++
;          ! Functional Description:
;          !       This global routine will search through the blast
;          !       table and look for non-zero locations and
;          !       then will blast tested array modules UV proms
;          !       with the new prom data.
;          !
;          ! Formal Parameters:
;          !       ARR$BNK_SEL:
;          !               Stores the array and bank select address.
;          !
;          ! Implicit Inputs:
;          !       BAD_BNK_REG, BLAST_TBL, FLG_REG, RET_STATUS
;          !
;          ! Implicit Outputs:
;          !       The data clock time out flag in the flag register
;          !       will be set if this control logic times out.
;          !
;          ! Completion codes:
;          !       none
;          !
;          ! Side Effects:
;          !       none
;          !--
;
;              label
;                  A;                                          !Leave loop label
;
;              local
;                  PROM_ADRS,                                  !Stores the generated prom address
;                  FINISH;
;
;              if .BNK_NUM_SEC eql 127 then FINISH = 127 else FINISH = 255;        !Is this a 16k part
;
;              WRT_PROM_MODE;                                  !Enable prom writes
;              PROM_ADRS = ZEROES;                             !Clear the prom adrs
;              PROM_ADRS<11, 4> = .ARR$BNK_SEL<.ARR_SEL_POS, ARR$SEL_SIZE>;        !Select the array
;
;              incr SEL_BNK from 0 to 3 do                     !Loop thru all the banks
;          A :
;                  begin
;
;                  if .BAD_BNK_REG [.SEL_BNK]                  !Is this a bad bank
;                  then
;                      begin
;                      PROM_ADRS<10, 1> = ZERO;                !Load the row address select bit
;                      PROM_ADRS<8, 2> = .SEL_BNK;             !Load the bank select
;
;                      incr ROW_NUM from 0 to .FINISH do       !Blast all the new row prom data
```

```
6578                    begin
6579
6580                    if .BLAST_TBL [.SEL_BNK, .ROW_NUM, FULL_WRD] neq ZERO   !Is this table entry not zero
6581                    then
6582                        begin
6583                        PROM_ADRS<0, 8> = .ROW_NUM; !Load the row number
6584                        WRT_RH (MLPA, PA_REG, .PROM_ADRS);  !Select the prom address
6585                        WRT_RH (MLPD, PD_REG, .BLAST_TBL [.SEL_BNK, .ROW_NUM, FULL_WRD]);   !Load the prom data
6586                        DAT_CLK;                        !Clock the new prom data in
6587                        DELAY (FIFTY_MS);               !Wait for the write to complete
6588
6589                        if .ML_ADDR [MLMR, D_CLK]   !Did the data clock time out
6590                        then
6591                            begin
6592                            ERRSF (ERR_10, TIME_OUT_MSG, 0);        !Report the error
6593                            PRINTB (ONE_MSG, REP_M7363_MSG);
6594                            PRINTB (ONE_MSG, SUPRES_MSG);
6595                            FLG_REG [F_D_CLK_TIME_OUT] = SET_FLG;
6596                            leave A;
6597                            end;
6598
6599                        end;
6600
6601                    end;
6602
6603                PROM_ADRS<10, 1> = ONE;             !Turn on the column addresses
6604
6605                incr COL_NUM from 0 to .FINISH do   !Blast all the new columns
6606                    begin
6607
6608                    if .BLAST_TBL [.SEL_BNK, .COL_NUM + .COL_BASE, FULL_WRD] neq ZERO
6609                                                    !Is this table ENTRY NOT zrro
6610                    then
6611                        begin
6612                        PROM_ADRS<0, 8> = .COL_NUM; !Load the column number
6613                        WRT_RH (MLPA, PA_REG, .PROM_ADRS);  !Select the prom address
6614                        WRT_RH (MLPD, PD_REG, .BLAST_TBL [.SEL_BNK, .COL_NUM + .COL_BASE, FULL_WRD]);
6615                        DAT_CLK;                        !Clock the new prom data in
6616                        DELAY (FIFTY_MS);               !Wait for the write to complete
6617
6618                        if .ML_ADDR [MLMR, D_CLK]   !Did the data clock time out
6619                        then
6620                            begin
6621                            ERRSF (ERR_10, TIME_OUT_MSG, 0);        !Report the error
6622                            PRINTB (ONE_MSG, REP_M7363_MSG);
6623                            PRINTB (ONE_MSG, SUPRES_MSG);
6624                            FLG_REG [F_D_CLK_TIME_OUT] = SET_FLG;
6625                            leave A;
6626                            end;
6627
6628                        end;
6629
```

```
;          6630                    end;
;          6631
;          6632                end;
;          6633
;          6634            end;
;          6635
;          6636        CLR_MBUS;                       !Clear the prom write enable
;          6637        DELAY (TEN_MS);                 !Wait for the voltage to settle
;          6638        end;


                                    .SBTTL  BL.PROMS ROUTINE DECLARATIONS
024636  004167  000000G         BL.PROMS:
                                    JSR     R1,$SAVE5                   ;                       6526
024642  162706  000034                SUB     #34,SP
024646  026727  040416' 000177        CMP     BNK.NUM.SEC,#177            ;                       6561
024654  001004                      BNE     1$
024656  012766  000177  000002      MOV     #177,2(SP)                  ; *,FINISH
024664  000403                      BR      2$
024666  012766  000377  000002  1$: MOV     #377,2(SP)                  ; *,FINISH
024674  016701  040374'          2$: MOV     ML.ADDR,R1
024700  016166  000024  000030      MOV     24(R1),30(SP)               ; *,ML.REG
024706  012702  000140              MOV     #140,R2                     ; *,MLREG
024712  016701  040374'              MOV     ML.ADDR,R1
024716  010261  000024              MOV     R2,24(R1)                   ; MLREG,*
024722  005003                      CLR     R3                          ; PROM.ADRS             6564
024724  012746  000054              MOV     #54,-(SP)                   ;                       6565
024730  060616                      ADD     SP,(SP)                     ; ARR$BNK.SEL,*
024732  016746  040412'              MOV     ARR.SEL.POS,-(SP)
024736  012746  000004              MOV     #4,-(SP)
024742  005046                      CLR     -(SP)
024744  004767  000000G             JSR     PC,BL$GT2
024750  000300                      SWAB    R0
024752  006300                      ASL     R0
024754  006300                      ASL     R0
024756  006300                      ASL     R0
024760  042700  103777              BIC     #103777,R0
024764  042703  074000              BIC     #74000,R3                   ; *,PROM.ADRS
024770  050003                      BIS     R0,R3                       ; *,PROM.ADRS
024772  005005                      CLR     R5                          ; SEL.BNK               6567
024774  010500                  3$: MOV     R5,R0                       ; SEL.BNK,*             6571
024776  006200                      ASR     R0
025000  006200                      ASR     R0
025002  006200                      ASR     R0
025004  062700  040420'              ADD     #BAD.BNK.REG,R0
025010  010046                      MOV     R0,-(SP)
025012  010546                      MOV     R5,-(SP)                    ; SEL.BNK,*
025014  042716  177770              BIC     #177770,(SP)
025020  012746  000001              MOV     #1,-(SP)
025024  005046                      CLR     -(SP)
025026  004767  000000G             JSR     PC,BL$GT2
025032  062706  000010              ADD     #10,SP
```

```
025036  006000                          ROR     R0
025040  103402                          BLO     4$
025042  000167  000626                  JMP     20$
025046  010504                  4$:     MOV     R5,R4                   ; SEL.BNK,*                     6575
025050  000304                          SWAB    R4
025052  042704  176377                  BIC     #176377,R4
025056  042703  003400                  BIC     #3400,R3                ; *,PROM.ADRS
025062  050403                          BIS     R4,R3                   ; *,PROM.ADRS
025064  010500                          MOV     R5,R0                   ; SEL.BNK,*                     6580
025066  000300                          SWAB    R0
025070  105000                          CLRB    R0
025072  006300                          ASL     R0
025074  010066  000010                  MOV     R0,10(SP)
025100  005004                          CLR     R4                      ; ROW.NUM                       6577
025102  000526                          BR      11$
025104  010400                  5$:     MOV     R4,R0                   ; ROW.NUM,*                     6580
025106  066600  000010                  ADD     10(SP),R0
025112  006300                          ASL     R0
025114  005760  030372'                 TST     BLAST.TBL(R0)
025120  001516                          BEQ     10$
025122  105003                          CLRB    R3                      ; PROM.ADRS                     6583
025124  150403                          BISB    R4,R3                   ; ROW.NUM,PROM.ADRS
025126  016701  040374'                 MOV     ML.ADDR,R1              ;                               6584
025132  016166  000020  000036          MOV     20(R1),36(SP)           ; *,ML.REG
025140  010302                          MOV     R3,R2                   ; PROM.ADRS,MLREG
025142  016701  040374'                 MOV     ML.ADDR,R1
025146  010261  000020                  MOV     R2,20(R1)               ; MLREG,*
025152  016701  040374'                 MOV     ML.ADDR,R1              ;                               6585
025156  016166  000046  000034          MOV     46(R1),34(SP)           ; *,ML.REG
025164  016002  030372'                 MOV     BLAST.TBL(R0),R2        ; *,MLREG
025170  016701  040374'                 MOV     ML.ADDR,R1
025174  010261  000046                  MOV     R2,46(R1)               ; MLREG,*
025200  016701  040374'                 MOV     ML.ADDR,R1
025204  016166  000024  000032          MOV     24(R1),32(SP)           ; *,ML.REG
025212  016602  000032                  MOV     32(SP),R2               ; ML.REG,MLREG
025216  152702  000020                  BISB    #20,R2                  ; *,MLREG
025222  016701  040374'                 MOV     ML.ADDR,R1
025226  010261  000024                  MOV     R2,24(R1)               ; MLREG,*
025232  012701  000144                  MOV     #144,R1                 ; *,$STMP2                      6587
025236  001411                  6$:     BEQ     9$
025240  016702  000000G                 MOV     LSDLY,R2                ; *,$STMP1
025244  001404                          BEQ     8$
025246  005066  000042          7$:     CLR     42(SP)                  ; $STMP
025252  005302                          DEC     R2                      ; $STMP1
025254  001374                          BNE     7$
025256  005301                  8$:     DEC     R1                      ; $STMP2
025260  000766                          BR      6$
025262  016701  040374'          9$:     MOV     ML.ADDR,R1              ;                               6589
025266  016166  000024  000030          MOV     24(R1),30(SP)           ; *,ML.REG
025274  032766  000020  000030          BIT     #20,30(SP)              ; *,ML.REG
025302  001425                          BEQ     10$
025304  104454                          TRAP    54                      ;                               6592
```

```
025306  000012                          .WORD   12
025310  002116'                         .WORD   TIME.OUT.MSG
025312  000000                          .WORD   0
025314  012746  002176'                 MOV     #REP.M7363.MSG,-(SP)            ;                               6593
025320  012746  002556'                 MOV     #ONE.MSG,-(SP)
025324  012746  000002                  MOV     #2,-(SP)
025330  010600                          MOV     SP,R0                           ; SP,*
025332  104414                          TRAP    14
025334  012716  001256'                 MOV     #SUPRES.MSG,(SP)                ;                               6594
025340  012746  002556'                 MOV     #ONE.MSG,-(SP)
025344  012746  000002                  MOV     #2,-(SP)
025350  010600                          MOV     SP,R0                           ; SP,*
025352  104414                          TRAP    14
025354  000535                          BR      17$                             ;                               6595
025356  005204                  10$:    INC     R4                              ; ROW.NUM                       6577
025360  020466  000012          11$:    CMP     R4,12(SP)                       ; ROW.NUM,FINISH
025364  003647                          BLE     5$
025366  052703  002000                  BIS     #2000,R3                        ; *,PROM.ADRS                   6603
025372  005004                          CLR     R4                              ; COL.NUM                       6605
025374  000534                          BR      19$
025376  010400                  12$:    MOV     R4,R0                           ; COL.NUM,*                     6608
025400  066700  040372'                 ADD     COL.BASE,R0
025404  066600  000010                  ADD     10(SP),R0
025410  006300                          ASL     R0
025412  005760  030372'                 TST     BLAST.TBL(R0)
025416  001522                          BEQ     18$
025420  105003                          CLRB    R3                              ; PROM.ADRS                     6612
025422  150403                          BISB    R4,R3                           ; COL.NUM,PROM.ADRS
025424  016701  040374'                 MOV     ML.ADDR,R1                      ;                               6613
025430  016166  000020  000026         MOV     20(R1),26(SP)                   ; *,ML.REG
025436  010302                          MOV     R3,R2                           ; PROM.ADRS,MLREG
025440  016701  040374'                 MOV     ML.ADDR,R1
025444  010261  000020                  MOV     R2,20(R1)                       ; MLREG,*
025450  016701  040374'                 MOV     ML.ADDR,R1                      ;                               6614
025454  016166  000046  000024         MOV     46(R1),24(SP)                   ; *,ML.REG
025462  016002  030372'                 MOV     BLAST.TBL(R0),R2                ; *,MLREG
025466  016701  040374'                 MOV     ML.ADDR,R1
025472  010261  000046                  MOV     R2,46(R1)                       ; MLREG,*
025476  016701  040374'                 MOV     ML.ADDR,R1
025502  016166  000024  000022         MOV     24(R1),22(SP)                   ; *,ML.REG
025510  016600  000022                  MOV     22(SP),R0                       ; ML.REG,MLREG
025514  152700  000020                  BISB    #20,R0                          ; *,MLREG
025520  016701  040374'                 MOV     ML.ADDR,R1
025524  010061  000024                  MOV     R0,24(R1)                       ; MLREG,*
025530  012702  000144                  MOV     #144,R2                         ; *,$STMP2                      6616
025534  001411                  13$:    BEQ     16$
025536  016701  000000G                 MOV     L$DLY,R1                        ; *,$STMP1
025542  001404                          BEQ     15$
025544  005066  000042          14$:    CLR     42(SP)                          ; $STMP
025550  005301                          DEC     R1                              ; $STMP1
025552  001374                          BNE     14$
025554  005302                  15$:    DEC     R2                              ; $STMP2
```

```
025556  000766                          BR      13$
025560  016701  040374'         16$:    MOV     ML.ADDR,R1                      ;                               6618
025564  016166  000024  000020         MOV     24(R1),20(SP)                   ; *,ML.REG
025572  031766  000020                  BIT     (PC),20(SP)                     ; *,ML.REG
025576  001432                          BEQ     18$
025600  104454                          TRAP    54                              ;                               6621
025602  000012                          .WORD   12
025604  002116'                         .WORD   TIME.OUT.MSG
025606  000000                          .WORD   0
025610  012746  002176'                 MOV     #REP.M7363.MSG,-(SP)            ;                               6622
025614  012746  002556'                 MOV     #ONE.MSG,-(SP)
025620  012746  000002                  MOV     #2,-(SP)
025624  010600                          MOV     SP,R0                           ; SP,*
025626  104414                          TRAP    14
025630  012716  001256'                 MOV     #SUPRES.MSG,(SP)                ;                               6623
025634  012746  002556'                 MOV     #ONE.MSG,-(SP)
025640  012746  000002                  MOV     #2,-(SP)
025644  010600                          MOV     SP,R0                           ; SP,*
025646  104414                          TRAP    14
025650  152767  000200  040422'  17$:   BISB    #200,FLG.REG                    ;                               6624
025656  062706  000012                  ADD     #12,SP                          ;                               6625
025662  000404                          BR      20$
025664  005204                  18$:    INC     R4                              ; COL.NUM                       6605
025666  020466  000012          19$:    CMP     R4,12(SP)                       ; COL.NUM,FINISH
025672  003641                          BLE     12$
025674  005205                  20$:    INC     R5                              ; SEL.BNK                       6567
025676  020527  000003                  CMP     R5,#3                           ; SEL.BNK,*
025702  003002                          BGT     21$
025704  000167  177064                  JMP     3$
025710  016701  040374'         21$:    MOV     ML.ADDR,R1                      ;                               6634
025714  016166  000010  000016         MOV     10(R1),16(SP)                   ; *,ML.REG
025722  016600  000016                  MOV     16(SP),R0                       ; ML.REG,MLREG
025726  152700  000040                  BISB    #40,R0                          ; *,MLREG
025732  016701  040374'                 MOV     ML.ADDR,R1
025736  010061  000010                  MOV     R0,10(R1)                       ; MLREG,*
025742  016701  040374'                 MOV     ML.ADDR,R1
025746  016166  000010  000014         MOV     10(R1),14(SP)                   ; *,ML.REG
025754  016600  000014                  MOV     14(SP),R0                       ; ML.REG,MLREG
025760  016705  040400'                 MOV     ML.DUT,R5
025764  042705  177770                  BIC     #177770,R5
025770  142700  000007                  BICB    #7,R0                           ; *,MLREG
025774  050500                          BIS     R5,R0                           ; *,MLREG
025776  016701  040374'                 MOV     ML.ADDR,R1
026002  010061  000010                  MOV     R0,10(R1)                       ; MLREG,*
026006  012702  023420                  MOV     #23420,R2                       ; *,$STMP2                      6637
026012  001411                  22$:    BEQ     25$
026014  016701  000000G                 MOV     L$DLY,R1                        ; *,$STMP1
026020  001404                          BEQ     24$
026022  005066  000042          23$:    CLR     42(SP)                          ; $STMP
026026  005301                          DEC     R1                              ; $STMP1
026030  001374                          BNE     23$
026032  005302                  24$:    DEC     R2                              ; $STMP2
```

```
026034  000766                          BR      22$
026036  062706  000044          25$:    ADD     #44,SP                  ;                       6526
026042  000207                          RTS     PC

; Routine Size:  323 words
; Maximum stack depth per invocation:  29 words
```

```
;          routine VER_BLAST (ARR$BNK_SEL) : novalue =
;              begin
;
;          !++
;          ! Functional Description:
;          !       After the proms have been blasted they
;          !       must be read back to insure the data was
;          !       written correctly.
;          !
;          !       This global routine reads the banks prom
;          !       data back and compares it to the respective
;          !       prom data stored in the error map.
;          !
;          !       On compare erors the operator will be notified
;          !       of a condition D and where the error occured.
;          !
;          ! Formal Parameters:
;          !       ARR$BNK_SEL:
;          !               Stores the array and bank select address.
;          !
;          ! Implicit Inputs:
;          !       ERROR_MAP, DEGRADE_MOD_REG, RET_STATUS
;          !
;          ! Implicit Outputs:
;          !       On compare errors this banks degrade mode register
;          !       bit is set.
;          !
;          ! Completion codes:
;          !       none
;          !
;          ! Side Effects:
;          !       none
;          !--
;
;              local
;                  PROM_ADRS,                          !Stores the generated prom address
;                  FINISH,                             !Variable ending sector number
;                  ARR_SEL;                            !Stores which array is selected for pm'ing
;
;              map
;                  ERROR_MAP : blockvector [4, 512] volatile;      !Make the error map look like the blast table
;
;              if .BNK_NUM_SEC eql 127 then FINISH = 127 else FINISH = 255;        !Is this a 16k chip
;
;              RD_PROM_MODE;                           !Enable prom read mode
;              ARR_SEL = .ARR$BNK_SEL<.ARR_SEL_POS, ARR$SEL_SIZE>; !Load the array select number
;              PROM_ADRS = ZEROES;                     !Clear the prom addres
;              PROM_ADRS<11, 4> = .ARR_SEL;            !Load  the array select number
;
;              incr SEL_BNK from 0 to 3 do             !Loop on all banks
;                  begin
;                  PROM_ADRS<10, 1> = ZERO;            !Select the row addresses
```

```
;          6691          PROM_ADRS<8, 2> = .SEL_BNK;                 !Select the bank
;          6692
;          6693          incr ROW_NUM from 0 to .FINISH do           !Look at all row addresses in the blast table
;          6694              begin
;          6695              PROM_ADRS<0, 8> = .ROW_NUM;             !Load the row number
;          6696              WRT_RH (MLPA, PA_REG, .PROM_ADRS);      !Select the prom address
;          6697              DELAY (ONE_US);                         !Wait for the prom data to come out
;          6698
;          6699              if .ERROR_MAP [.SEL_BNK, .ROW_NUM, FULL_WRD] neq .ML_ADDR [MLPD, PD_REG]
;          6700              then
;          6701                  begin
;          6702                  ERRSF (ERR_8, CON_D_MSG, 0);        !Report the error
;          6703                  PRINTB (A_B_PRINT, .ARR_SEL, .SEL_BNK);
;          6704                  DEGRADE_MOD_REG [.SEL_BNK] = SET_FLG;
;          6705                  end;
;          6706
;          6707              end;
;          6708
;          6709          PROM_ADRS<10, 1> = ONE;                     !Select the column addresses
;          6710
;          6711          incr COL_NUM from 0 to .FINISH do           !Look at all the columns
;          6712              begin
;          6713              PROM_ADRS<0, 8> = .COL_NUM;             !Select the column addresses
;          6714              WRT_RH (MLPA, PA_REG, .PROM_ADRS);      !Select the prom address
;          6715              DELAY (ONE_US);                         !Wait for the prom data to come out
;          6716
;          6717              if .ERROR_MAP [.SEL_BNK, .COL_NUM + .COL_BASE, FULL_WRD] neq .ML_ADDR [MLPD, PD_REG]
;          6718              then
;          6719                  begin
;          6720                  ERRSF (ERR_8, CON_D_MSG, 0);        !Report the error
;          6721                  PRINTB (A_B_PRINT, .ARR_SEL, .SEL_BNK);
;          6722                  DEGRADE_MOD_REG [.SEL_BNK] = SET_FLG;
;          6723                  end;
;          6724
;          6725              end;
;          6726
;          6727          end;
;          6728
;          6729      CLR_MBUS;                                       !Clear out the prom read mode
;          6730      end;

                                .SBTTL  VER.BLAST ROUTINE DECLARATIONS
026044  004167  000000G       VER.BLAST:
                                JSR     R1,$SAVE5                   ;                   6639
026050  162706  000024          SUB     #24,SP
026054  026727  040416' 000177  CMP     BNK.NUM.SEC,#177            ;                   6681
026062  001003                  BNE     1$
026064  012716  000177          MOV     #177,(SP)                   ; *,FINISH
026070  000402                  BR      2$
026072  012716  000377  1$:     MOV     #377,(SP)                   ; *,FINISH
026076  016701  040374'  2$:     MOV     ML.ADDR,R1
```

```
026102  016166  000024  000020          MOV     24(R1),20(SP)                   ; *,ML.REG
026110  012700  000040                  MOV     #40,R0                          ; *,MLREG
026114  016701  040374'                 MOV     ML.ADDR,R1
026120  010061  000024                  MOV     R0,24(R1)                       ; MLREG,*
026124  012746  000044                  MOV     #44,-(SP)                       ;                                       6684
026130  060616                          ADD     SP,(SP)                         ; ARR$BNK.SEL,*
026132  016746  040412'                 MOV     ARR.SEL.POS,-(SP)
026136  012746  000004                  MOV     #4,-(SP)
026142  005046                          CLR     -(SP)
026144  004767  000000G                 JSR     PC,BL$GT2
026150  010005                          MOV     R0,R5                           ; *,ARR.SEL
026152  005004                          CLR     R4                              ; PROM.ADRS                             6685
026154  010503                          MOV     R5,R3                           ; ARR.SEL,*                             6686
026156  000303                          SWAB    R3
026160  006303                          ASL     R3
026162  006303                          ASL     R3
026164  006303                          ASL     R3
026166  042703  103777                  BIC     #103777,R3
026172  042704  074000                  BIC     #74000,R4                       ; *,PROM.ADRS
026176  050304                          BIS     R3,R4                           ; *,PROM.ADRS
026200  005002                          CLR     R2                              ; SEL.BNK                               6688
026202  010203                  3$:     MOV     R2,R3                           ; SEL.BNK,*                             6691
026204  000303                          SWAB    R3
026206  042703  176377                  BIC     #176377,R3
026212  042704  003400                  BIC     #3400,R4                        ; *,PROM.ADRS
026216  050304                          BIS     R3,R4                           ; *,PROM.ADRS
026220  010200                          MOV     R2,R0                           ; SEL.BNK,*                             6699
026222  000300                          SWAB    R0
026224  105000                          CLRB    R0
026226  006300                          ASL     R0
026230  010066  000012                  MOV     R0,12(SP)
026234  005003                          CLR     R3                              ; ROW.NUM                               6693
026236  000503                          BR      10$
026240  105004                  4$:     CLRB    R4                              ; PROM.ADRS                             6695
026242  150304                          BISB    R3,R4                           ; ROW.NUM,PROM.ADRS
026244  016701  040374'                 MOV     ML.ADDR,R1                      ;                                       6696
026250  016166  000020  000026          MOV     20(R1),26(SP)                   ; *,ML.REG
026256  010400                          MOV     R4,R0                           ; PROM.ADRS,MLREG
026260  016701  040374'                 MOV     ML.ADDR,R1
026264  010061  000020                  MOV     R0,20(R1)                       ; MLREG,*
026270  012700  000001                  MOV     #1,R0                           ; *,$STMP2                              6697
026274  001411                  5$:     BEQ     8$
026276  016701  000000G                 MOV     L$DLY,R1                        ; *,$STMP1
026302  001404                          BEQ     7$
026304  005066  000032          6$:     CLR     32(SP)                          ; $STMP
026310  005301                          DEC     R1                              ; $STMP1
026312  001374                          BNE     6$
026314  005300                  7$:     DEC     R0                              ; $STMP2
026316  000766                          BR      5$
026320  010300                  8$:     MOV     R3,R0                           ; ROW.NUM,*                             6699
026322  066600  000012                  ADD     12(SP),R0
026326  006300                          ASL     R0
```

```
026330  016701  040374'                        MOV     ML.ADDR,R1
026334  016166  000046  000024                MOV     46(R1),24(SP)                   ; *,ML.REG
026342  026066  010230' 000024                CMP     ERROR.MAP(R0),24(SP)            ; *,ML.REG
026350  001435                                BEQ     9$
026352  104454                                TRAP    54                              ;                                       6702
026354  000010                                .WORD   10
026356  001436'                               .WORD   CON.D.MSG
026360  000000                                .WORD   0
026362  010246                                MOV     R2,-(SP)                        ; SEL.BNK,*                             6703
026364  010546                                MOV     R5,-(SP)                        ; ARR.SEL,*
026366  012746  003264'                        MOV     #A.B.PRINT,-(SP)
026372  012746  000003                        MOV     #3,-(SP)
026376  010600                                MOV     SP,R0                           ; SP,*
026400  104414                                TRAP    14
026402  010200                                MOV     R2,R0                           ; SEL.BNK,*                             6704
026404  006200                                ASR     R0
026406  006200                                ASR     R0
026410  006200                                ASR     R0
026412  062700  040424'                        ADD     #DEGRADE.MOD.REG,R0
026416  010016                                MOV     R0,(SP)
026420  010246                                MOV     R2,-(SP)                        ; SEL.BNK,*
026422  042716  177770                        BIC     #177770,(SP)
026426  012746  000001                        MOV     #1,-(SP)
026432  011646                                MOV     (SP),-(SP)
026434  004767  000000G                       JSR     PC,BL$PU2
026440  062706  000016                        ADD     #16,SP                          ;                                       6701
026444  005203                      9$:       INC     R3                              ; ROW.NUM                               6693
026446  020366  000010                10$:    CMP     R3,10(SP)                       ; ROW.NUM,FINISH
026452  003672                                BLE     4$
026454  052704  002000                        BIS     #2000,R4                        ; *,PROM.ADRS                           6709
026460  005003                                CLR     R3                              ; COL.NUM                               6711
026462  000505                                BR      17$
026464  105004                      11$:    CLRB    R4                              ; PROM.ADRS                             6713
026466  150304                                BISB    R3,R4                           ; COL.NUM,PROM.ADRS
026470  016701  040374'                        MOV     ML.ADDR,R1                      ;                                       6714
026474  016166  000020  000022                MOV     20(R1),22(SP)                   ; *,ML.REG
026502  010400                                MOV     R4,R0                           ; PROM.ADRS,MLREG
026504  016701  040374'                        MOV     ML.ADDR,R1
026510  010061  000020                        MOV     R0,20(R1)                       ; MLREG,*
026514  012700  000001                        MOV     #1,R0                           ; *,$STMP2                              6715
026520  001411                      12$:    BEQ     15$
026522  016701  000000G                       MOV     L$DLY,R1                        ; *,$STMP1
026526  001404                                BEQ     14$
026530  005066  000032                13$:    CLR     32(SP)                          ; $STMP
026534  005301                                DEC     R1                              ; $STMP1
026536  001374                                BNE     13$
026540  005300                      14$:    DEC     R0                              ; $STMP2
026542  000766                                BR      12$
026544  010300                      15$:    MOV     R3,R0                           ; COL.NUM,*                             6717
026546  066700  040372'                        ADD     COL.BASE,R0
026552  066600  000012                        ADD     12(SP),R0
026556  006300                                ASL     R0
```

```
026560  016701  040374'                        MOV     ML.ADDR,R1
026564  016166  000046  000020                MOV     46(R1),20(SP)                   ; *,ML.REG
026572  026066  010230' 000020                CMP     ERROR.MAP(R0),20(SP)            ; *,ML.REG
026600  001435                                BEQ     16$
026602  104454                                TRAP    54                              ;                                       6720
026604  000010                                .WORD   10
026606  001436'                               .WORD   CON.D.MSG
026610  000000                                .WORD   0
026612  010246                                MOV     R2,-(SP)                        ; SEL.BNK,*                             6721
026614  010546                                MOV     R5,-(SP)                        ; ARR.SEL,*
026616  012746  003264'                       MOV     #A.B.PRINT,-(SP)
026622  012746  000003                        MOV     #3,-(SP)
026626  010600                                MOV     SP,R0                           ; SP,*
026630  104414                                TRAP    14
026632  010200                                MOV     R2,R0                           ; SEL.BNK,*                             6722
026634  006200                                ASR     R0
026636  006200                                ASR     R0
026640  006200                                ASR     R0
026642  062700  040424'                       ADD     #DEGRADE.MOD.REG,R0
026646  010016                                MOV     R0,(SP)
026650  010246                                MOV     R2,-(SP)                        ; SEL.BNK,*
026652  042716  177770                        BIC     #177770,(SP)
026656  012746  000001                        MOV     #1,-(SP)
026662  011646                                MOV     (SP),-(SP)
026664  004767  000000G                       JSR     PC,BL$PU2
026670  062706  000016                        ADD     #16,SP                          ;                                       6719
026674  005203                          16$:  INC     R3                              ; COL.NUM                               6711
026676  020366  000010                  17$:  CMP     R3,10(SP)                       ; COL.NUM,FINISH
026702  003670                                BLE     11$
026704  005202                                INC     R2                              ; SEL.BNK                               6688
026706  020227  000003                        CMP     R2,#3                           ; SEL.BNK,*
026712  003002                                BGT     18$
026714  000167  177262                        JMP     3$
026720  016701  040374'                  18$:  MOV     ML.ADDR,R1                      ;                                       6727
026724  016166  000010  000016               MOV     10(R1),16(SP)                   ; *,ML.REG
026732  016600  000016                        MOV     16(SP),R0                       ; ML.REG,MLREG
026736  152700  000040                        BISB    #40,R0                          ; *,MLREG
026742  016701  040374'                       MOV     ML.ADDR,R1
026746  010061  000010                        MOV     R0,10(R1)                       ; MLREG,*
026752  016701  040374'                       MOV     ML.ADDR,R1
026756  016166  000010  000014               MOV     10(R1),14(SP)                   ; *,ML.REG
026764  016600  000014                        MOV     14(SP),R0                       ; ML.REG,MLREG
026770  016705  040400'                       MOV     ML.DUT,R5
026774  042705  177770                        BIC     #177770,R5
027000  142700  000007                        BICB    #7,R0                           ; *,MLREG
027004  050500                                BIS     R5,R0                           ; *,MLREG
027006  016701  040374'                       MOV     ML.ADDR,R1
027012  010061  000010                        MOV     R0,10(R1)                       ; MLREG,*
027016  062706  000034                        ADD     #34,SP                          ;                                       6639
027022  000207                                RTS     PC

; Routine Size:  248 words
```

```
; Maximum stack depth per invocation:  27 words
```

```
;       6731  routine VER_ERROR_MASK (ARR$BNK_SEL, TSTED_BNK) : novalue =
;       6732      begin
;       6733
;       6734  !++
;       6735  ! Functional Description:
;       6736  !       After the proms have been blasted and the proms are read for a good blast the
;       6737  !       selected array is write checked again and made sure that all blasted rows and
;       6738  !       columns have been successfully blasted.
;       6739  !
;       6740  !       UNC errors are not tolerated in any bank and warrents the array to be replaced.
;       6741  !       single chip failures in degrade mode banks is tolerated and does not constiture an error.
;       6742  !       single chip failures in non-degrade node banks is not tolerated and will cause an
;       6743  !       error and that the array be replaced.
;       6744  !
;       6745  ! Formal Parameters:
;       6746  !       ARR$BNK_SEL:
;       6747  !               Stores the array and bank select address.
;       6748  !
;       6749  ! Implicit Inputs:
;       6750  !       BAD_BNK_REG, FLG_REG, DEGRADE_MOD_REG
;       6751  !
;       6752  ! Implicit Outputs:
;       6753  !       none
;       6754  !
;       6755  ! Completion codes:
;       6756  !       none
;       6757  !
;       6758  ! Side Effects:
;       6759  !       none
;       6760  !--
;       6761
;       6762      local
;       6763          BNK_NUM,                                !Stores the selected bank number
;       6764          ARR_NUM;                                !Stores the selected array number
;       6765
;       6766      ARR_NUM = .ARR$BNK_SEL<.ARR_SEL_POS, ARR$SEL_SIZE>; !Load the array number
;       6767
;       6768      incr SEL_BNK from 0 to .TSTED_BNK - 1 do    !Verify all banks
;       6769          begin
;       6770          BNK_NUM = .ARR$BNK_SEL<.BNK_SEL_POS, BNK$SEL_SIZE>;     !Load the bank sel number
;       6771
;       6772          if .BAD_BNK_REG [.BNK_NUM]              !Is this a bad bank
;       6773          then
;       6774              begin
;       6775              FB_CHIPS (.ARR$BNK_SEL);            !Find any bad bits
;       6776
;       6777              if S_CHIP_TBL (ZERO) geq ZERO       !Were any bad bits found
;       6778              then
;       6779                  begin
;       6780
;       6781                  if .FLG_REG [F_UNC_ERR_FLG]     !Was it an unc error
;       6782                  then
```

```
;          6783                         begin
;          6784                         ERRSF (ERR_11, UNC_ERR_MSG, 0);     !Report the error
;          6785                         PRINTB (REP_PRINT, .ARR_NUM);
;          6786                         end
;          6787                     else
;          6788                         begin
;          6789
;          6790                         if not .DEGRADE_MOD_REG [.BNK_NUM]  !Is this not a degrade mode bank
;          6791                         then
;          6792                             begin
;          6793                             ERRSF (ERR_12, MEM_ERR_MSG, 0); !Report the error
;          6794                             PRINTB (REP_PRINT, .ARR_NUM);
;          6795                             end;
;          6796
;          6797                         end;
;          6798
;          6799                     end;
;          6800
;          6801                 end;
;          6802
;          6803             ARR$BNK_SEL = .ARR$BNK_SEL + .INC_BNK;  !Select the next bank
;          6804             end;
;          6805
;          6806         end;


                                        .SBTTL  VER.ERROR.MASK ROUTINE DECLARATIONS
027024  004167  000000G          VER.ERROR.MASK:
                                        JSR     R1,$SAVE4                       ;                                   6731
027030  012746  000020                  MOV     #20,-(SP)                       ;                                   6766
027034  060616                          ADD     SP,(SP)                         ; ARR$BNK.SEL,*
027036  016746  040412'                 MOV     ARR.SEL.POS,-(SP)
027042  012746  000004                  MOV     #4,-(SP)
027046  005046                          CLR     -(SP)
027050  004767  000000G                 JSR     PC,BL$GT2
027054  010004                          MOV     R0,R4                           ; *,ARR.NUM
027056  005001                          CLR     R1                              ; SEL.BNK                           6768
027060  000533                          BR      6$
027062  012746  000030          1$:     MOV     #30,-(SP)                       ;                                   6770
027066  060616                          ADD     SP,(SP)                         ; ARR$BNK.SEL,*
027070  016746  040414'                 MOV     BNK.SEL.POS,-(SP)
027074  012746  000002                  MOV     #2,-(SP)
027100  005046                          CLR     -(SP)
027102  004767  000000G                 JSR     PC,BL$GT2
027106  010002                          MOV     R0,R2                           ; *,BNK.NUM
027110  010203                          MOV     R2,R3                           ; BNK.NUM,*                         6772
027112  006203                          ASR     R3
027114  006203                          ASR     R3
027116  006203                          ASR     R3
027120  010346                          MOV     R3,-(SP)
027122  062716  040420'                 ADD     #BAD.BNK.REG,(SP)
027126  010246                          MOV     R2,-(SP)                        ; BNK.NUM,*
```

```
027130  042716  177770                          BIC     #177770,(SP)
027134  012746  000001                          MOV     #1,-(SP)
027140  005046                                  CLR     -(SP)
027142  004767  000000G                         JSR     PC,BL$GT2
027146  062706  000010                          ADD     #10,SP
027152  006000                                  ROR     R0
027154  103067                                  BCC     5$
027156  016646  000036                          MOV     36(SP),-(SP)            ; ARR$BNK.SEL,*                 6775
027162  004767  174206                          JSR     PC,FB.CHIPS
027166  005046                                  CLR     -(SP)                   ;                               6777
027170  004767  165114                          JSR     PC,S.CHIP.TBL
027174  005726                                  TST     (SP)+
027176  005700                                  TST     R0
027200  002454                                  BLT     4$
027202  032767  000002  040422'                 BIT     #2,FLG.REG              ;                               6781
027210  001414                                  BEQ     2$
027212  104454                                  TRAP    54                      ;                               6784
027214  000013                                  .WORD   13
027216  002040'                                 .WORD   UNC.ERR.MSG
027220  000000                                  .WORD   0
027222  010446                                  MOV     R4,-(SP)                ; ARR.NUM,*                     6785
027224  012746  003200'                         MOV     #REP.PRINT,-(SP)
027230  012746  000002                          MOV     #2,-(SP)
027234  010600                                  MOV     SP,R0                   ; SP,*
027236  104414                                  TRAP    14
027240  000432                                  BR      3$                      ;                               6783
027242  010346                          2$:     MOV     R3,-(SP)                ;                               6790
027244  062716  040424'                         ADD     #DEGRADE.MOD.REG,(SP)
027250  010246                                  MOV     R2,-(SP)                ; BNK.NUM,*
027252  042716  177770                          BIC     #177770,(SP)
027256  012746  000001                          MOV     #1,-(SP)
027262  005046                                  CLR     -(SP)
027264  004767  000000G                         JSR     PC,BL$GT2
027270  062706  000010                          ADD     #10,SP
027274  006000                                  ROR     R0
027276  103415                                  BLO     4$
027300  104454                                  TRAP    54                      ;                               6793
027302  000014                                  .WORD   14
027304  002236'                                 .WORD   MEM.ERR.MSG
027306  000000                                  .WORD   0
027310  010446                                  MOV     R4,-(SP)                ; ARR.NUM,*                     6794
027312  012746  003200'                         MOV     #REP.PRINT,-(SP)
027316  012746  000002                          MOV     #2,-(SP)
027322  010600                                  MOV     SP,R0                   ; SP,*
027324  104414                                  TRAP    14
027326  062706  000006                  3$:     ADD     #6,SP                   ;                               6792
027332  005726                          4$:     TST     (SP)+                   ;                               6774
027334  066766  040406'  000036        5$:     ADD     INC.BNK,36(SP)          ; *,ARR$BNK.SEL                 6803
027342  062706  000010                          ADD     #10,SP                  ;                               6769
027346  005201                                  INC     R1                      ; SEL.BNK                       6768
027350  020166  000024                  6$:     CMP     R1,24(SP)               ; SEL.BNK,TSTED.BNK
027354  002642                                  BLT     1$
```

```
027356  062706  000010                  ADD     #10,SP                          ;                       6732
027362  000207                          RTS     PC                              ;                       6731

; Routine Size:  112 words
; Maximum stack depth per invocation:  18 words
```

```
;       6807  %sbttl 'INITIALIZE CODE SECTION'
;       6808  !
;       6809  ! INITIALIZE CODE SECTION
;       6810  !
;       6811  BGNINIT;
;       6812
;       6813  local
;       6814      !
;       6815      !Hardware ptable entry temporary storage locations
;       6816      !
;       6817      sys_hwtbl,                                  !prom maint the entire system
;       6818      ARR_HWTBL,                                  !Prom maint a single array module
;       6819      BNK_HWTBL,                                  !Prom maint a single bank
;       6820      BNK_NO_HWTBL,                               !Bank number to be prom maint
;       6821      ARR_NO_HWTBL,                               !Array module number to be prom maint
;       6822      RH_BASE_HWTBL,                              !Rh base register address
;       6823      DUT_HWTBL,                                  !Divice number to be prom maint
;       6824      OPTION_HWTBL,                               !Drive option code 1 = ml-11a, 0 = ml-11b
;       6825      CORRECT_HWTBL,                              !Are the program parameters correct flag
;       6826      PTBL_PTR,                                   !Hardware ptable pointer
;       6827      LUN;                                        !Supervisor logical unit number variable
;       6828
;       6829  if .L$UNIT gtr ONE                            !Where more one ptable built
;       6830  then
;       6831      begin
;       6832      ERRSF (ERR_9, HQ_ERR_MSG, 0);               !Report the error
;       6833      PRINTB (ONE_MSG, GTR_MSG);                  !Report what the error is
;       6834      PRINTB (ONE_MSG, UNIT_SEL_MSG);             !Report that the first ptable built will be used
;       6835      end;
;       6836
;       6837  L$UNIT = 0;                                    !Only one drive is to be tested
;       6838  !
;       6839  ! Because of the inpact of this program on the
;       6840  ! ML-11 system only the start command will be
;       6841  ! recognized to start program execution
;       6842  !
;       6843
;       6844  if not (READEF (EF_START))                     !Was the start command used to start the program
;       6845  then
;       6846      begin
;       6847      ERRSF (ERR_1, ILL_CMD_MSG, 0);              !Report the error
;       6848      PRINTB (ONE_MSG, START_MSG);                !Report what the error is
;       6849      DODU (.L$UNIT);                             !Drop this unit
;       6850      DOCLN;                                      !Go to the clean up code section
;       6851      end;
;       6852
;       6853  !
;       6854  ! The contents of the first p-table built (lun 0 )
;       6855  ! will be the selected progran parameters.  all other remaining
;       6856  ! P-tables built will be ignored.
;       6857  !
;       6858  LUN = ZERO;
```

```
6859
6860  if (GPHARD (.LUN, PTBL_PTR)) eql ZERO           !Is the first ptable present
6861  then
6862      begin
6863      ERRSF (ERR_3, LUN_MISS_MSG, 0);             !Report the error
6864      PRINTB (ONE_MSG, SUPRES_MSG);               !Report suppress message
6865      PRINTB (ONE_MSG, RET_DRS_MSG);              !Report returning to drs>
6866      DODU (.L$UNIT);                             !Drop this unit
6867      DOCLN;                                      !Go to the drop unit code section
6868      end
6869  else
6870      begin
6871      !
6872      ! Input to the program the hardware ptable
6873      ! entries and store them into the temporary
6874      ! storage locations
6875      !
6876      SYS_HWTBL = .((.PTBL_PTR) + %o'0');
6877      ARR_HWTBL = .((.PTBL_PTR) + %o'2');
6878      BNK_HWTBL = .((.PTBL_PTR) + %o'4');
6879      BNK_NO_HWTBL = .((.PTBL_PTR) + %o'6');
6880      ARR_NO_HWTBL = .((.PTBL_PTR) + %o'10');
6881      RH_BASE_HWTBL = .((.PTBL_PTR) + %o'12');
6882      DUT_HWTBL = .((.PTBL_PTR) + %o'14');
6883      OPTION_HWTBL = .((.PTBL_PTR) + %o'16');
6884      CORRECT_HWTBL = .((.PTBL_PTR) + %o'20');
6885      end;
6886
6887  !
6888  ! Before we start, lets see if the operator
6889  ! entered his parameters correctly.
6890  !
6891
6892  if not .CORRECT_HWTBL                           !Are the inputed parameters correct
6893  then
6894      begin
6895      PRINTB (LFS);                               !Print some line feeds to be neat
6896      PRINTB (ONE_MSG, SUPRES_MSG);               !Print the suppress message
6897      PRINTB (ONE_MSG, RET_DRS_MSG);              !Print returning to drs> message
6898      DODU (.L$UNIT);                             !Drop this unit
6899      DOCLN;                                      !Go to the clean up code section
6900      end
6901  else
6902      begin
6903      !
6904      ! Build the run time parameters
6905      !
6906      arr$bnk_sel = zeroes;                       !Load the array & bank select variable with zeroes
6907      ML_ADDR = .RH_BASE_HWTBL;                   !Load the ml-11 base address into ml_addr
6908      ML_DUT = .DUT_HWTBL;                        !Load ml_dut with the divices drive select number
6909      !
6910      ! Set up 16k or 64k mos ram run time parameters
```

```
;         6911      !
;         6912
;         6913      if .OPTION_HWTBL eql ML11A                    !Is this a ml-11a option 16k parts
;         6914      then
;         6915          begin
;         6916          COL_BASE = 128;                           !Load the column base offset
;         6917          INC_BNK = %o'200';                        !Load the bank increment variable
;         6918          INC_ARR = %o'1000';                       !Load the array increment variable
;         6919          BNK_NUM_SEC = 127;                        !Load the banks number of sectors variable
;         6920          ARR_SEL_POS = 9;                          !Load the array select position constant
;         6921          BNK_SEL_POS = 7;                          !Loat the bank select position constant
;         6922          MAX_CHIP_COL = 128;                       !Load the maximum columns per chip variable
;         6923          end
;         6924      else
;         6925          begin
;         6926          COL_BASE = 256;                           !Load the column base offset variable
;         6927          INC_BNK = %o'1000';                       !Load the bank increment variable
;         6928          INC_ARR = %o'4000';                       !Load the array increment variable
;         6929          BNK_NUM_SEC = 511;                        !Load the banks number of sectors variable
;         6930          ARR_SEL_POS = 11;                         !Load the array selection position constant
;         6931          BNK_SEL_POS = 9;                          !Load the bank selection position constant
;         6932          MAX_CHIP_COL = 256;                       !Load the maximum columns per chip variable
;         6933          end;
;         6934
;         6935      !
;         6936      ! Build the run time parameter for masking
;         6937      ! either the entire system, an entire array or
;         6938      ! a single bank.
;         6939      !
;         6940
;         6941      if .SYS_HWTBL                                 !Is the entire ml-11 system to be prom maint
;         6942      then
;         6943          begin
;         6944          WRT_RH (MLCS2, DRV_SEL, .ML_DUT);         !Select the drive under test
;         6945          LST_TSTED_ARR = .ML_ADDR [MEMR, SIZING] - 1;    !Read and save the sizing
;         6946          RAND_PASS = 4;                            !Do two random data passes
;         6947          TSTED_BNK = 4;                            !Test four banks per array
;         6948          end
;         6949      else
;         6950          begin
;         6951
;         6952          if .ARR_HWTBL                             !Is a single array to be prom maint
;         6953          then
;         6954              begin
;         6955              LST_TSTED_ARR = 0;                    !Load loop variable to test only one array
;         6956              RAND_PASS = 8;                        !Do three passes of random data
;         6957              TSTED_BNK = 4;                        !Test all four banks on the array
;         6958              ARR$BNK_SEL<.ARR_SEL_POS, ARR$SEL_SIZE> = .ARR_NO_HWTBL;    !Select which array to test
;         6959              end
;         6960          else
;         6961              begin
;         6962
```

```
;          6963                if .BNK_HWTBL                           !Is a single bank to be prom maint
;          6964                then
;          6965                    begin
;          6966                    LST_TSTED_ARR = 0;                  !Load loop variable to test only one array
;          6967                    RAND_PASS = 13;                     !Do five passes of random data
;          6968                    TSTED_BNK = 1;                      !Test only one bank
;          6969                    ARR$BNK_SEL<.ARR_SEL_POS, ARR$SEL_SIZE> = .ARR_NO_HWTBL;
;          6970                                                        !Select which array to prom maint
;          6971                    ARR$BNK_SEL<.BNK_SEL_POS, BNK$SEL_SIZE> = .BNK_NO_HWTBL;
;          6972                                                        !Select which bank to prom maint
;          6973                    end
;          6974                else
;          6975                    begin
;          6976                    !
;          6977                    ! The hardware questions were not answered
;          6978                    ! correctly.  Masking either a system, array
;          6979                    ! or bank was not selected.  Report the error
;          6980                    ! and rerturn to DRS>.
;          6981                    !
;          6982                    ERRSF (ERR_2, HQ_ERR_MSG, 0);       !Report the error
;          6983                    PRINTB (ONE_MSG, HQ_MSG);           !Report what the error is
;          6984                    PRINTB (ONE_MSG, SUPRES_MSG);       !Print suppress message
;          6985                    PRINTB (ONE_MSG, RET_DRS_MSG);      !Print returning to drs> message
;          6986                    DODU (.L$UNIT);                     !Drop this unit
;          6987                    DOCLN;                              !Go to the clean up code section
;          6988                    end;
;          6989
;          6990                end;
;          6991
;          6992            end;
;          6993
;          6994        end;
;          6995    !
;          6996    ! VER CZMLCB ADDED PM_COUNT
;          6997    !
;          6998    PM_COUNT = 0;                                       !Init the prom maint counter
;          6999
;          7000    !+
;          7001    !
;          7002    ! Added this init code of this structure here.
;          7003    !
;          7004    ! The prom maint error log table stores the number of
;          7005    ! detected all bad rows and or columns detected during
;          7006    ! the run time of this diagnostic. In this table is stored
;          7007    ! the : unit #, board #, bank #, bit #
;          7008    ! and a count of how many all bad rows and or columns
;          7009    ! detected withib each chip.
;          7010    !
;          7011    ! This code initializes this structure to zeroes
;          7012    ! before starting execution of the exerciser.
;          7013    !-
;          7014
```

```
;         7015  incr index from 0 to 127 do                        !Clear the prom maint error log table
;         7016      begin
;         7017      PM_LOG [.index, WRD$_0] = ZERO;                !Clear word zero
;         7018      PM_LOG [.index, WRD$_1] = ZERO;                !Clear word one
;         7019      end;
;         7020
;         7021  ENDINIT;


                                        .SBTTL  LINIT INITIALIZE CODE SECTION
027364  004167  000000G            LINIT: JSR     R1,$SAVE5                        ;                                  6806
027370  162706  000014                   SUB     #14,SP
027374  026727  000000G 000001           CMP     L$UNIT,#1                        ;                                  6829
027402  003426                           BLE     1$
027404  104454                           TRAP    54                               ;                                  6832
027406  000011                           .WORD   11
027410  001070'                          .WORD   HQ.ERR.MSG
027412  000000                           .WORD   0
027414  012746  001506'                  MOV     #GTR.MSG,-(SP)                   ;                                  6833
027420  012746  002556'                  MOV     #ONE.MSG,-(SP)
027424  012746  000002                   MOV     #2,-(SP)
027430  010600                           MOV     SP,R0                            ; SP,*
027432  104414                           TRAP    14
027434  012716  001562'                  MOV     #UNIT.SEL.MSG,(SP)               ;                                  6834
027440  012746  002556'                  MOV     #ONE.MSG,-(SP)
027444  012746  000002                   MOV     #2,-(SP)
027450  010600                           MOV     SP,R0                            ; SP,*
027452  104414                           TRAP    14
027454  062706  000012                   ADD     #12,SP                           ;                                  6831
027460  005067  000000G            1$:    CLR     L$UNIT                           ;                                  6837
027464  012700  000040                   MOV     #40,R0                           ;                                  6844
027470  104447                           TRAP    47
027472  103422                           BCS     2$
027474  104454                           TRAP    54                               ;                                  6847
027476  000001                           .WORD   1
027500  000726'                          .WORD   ILL.CMD.MSG
027502  000000                           .WORD   0
027504  012746  001032'                  MOV     #START.MSG,-(SP)                 ;                                  6848
027510  012746  002556'                  MOV     #ONE.MSG,-(SP)
027514  012746  000002                   MOV     #2,-(SP)
027520  010600                           MOV     SP,R0                            ; SP,*
027522  104414                           TRAP    14
027524  016700  000000G                  MOV     L$UNIT,R0                        ;                                  6849
027530  104451                           TRAP    51
027532  104444                           TRAP    44
027534  062706  000006                   ADD     #6,SP                            ;                                  6846
027540  005000                     2$:   CLR     R0                               ; LUN                              6858
027542  104442                           TRAP    42                               ;                                  6860
027544  010005                           MOV     R0,R5                            ; *,PTBL.PTR
027546  001033                           BNE     3$
027550  104454                           TRAP    54                               ;                                  6863
027552  000003                           .WORD   3
```

```
027554  001340'                         .WORD   LUN.MISS.MSG
027556  000000                          .WORD   0
027560  012746  001256'                MOV     #SUPRES.MSG,-(SP)               ;                                       6864
027564  012746  002556'                MOV     #ONE.MSG,-(SP)
027570  012746  000002                  MOV     #2,-(SP)
027574  010600                          MOV     SP,R0                           ; SP,*
027576  104414                          TRAP    14
027600  012716  001316'                MOV     #RET.DRS.MSG,(SP)               ;                                       6865
027604  012746  002556'                MOV     #ONE.MSG,-(SP)
027610  012746  000002                  MOV     #2,-(SP)
027614  010600                          MOV     SP,R0                           ; SP,*
027616  104414                          TRAP    14
027620  016700  000000G                 MOV     L$UNIT,R0                       ;                                       6866
027624  104451                          TRAP    51
027626  104444                          TRAP    44
027630  062706  000012                  ADD     #12,SP                          ;                                       6862
027634  000424                          BR      4$                              ;                                       6860
027636  011566  000006          3$:     MOV     (R5),6(SP)                      ; PTBL.PTR,SYS.HWTBL                    6876
027642  016566  000002  000004         MOV     2(R5),4(SP)                     ; *(PTBL.PTR),ARR.HWTBL                 6877
027650  016566  000004  000002         MOV     4(R5),2(SP)                     ; *(PTBL.PTR),BNK.HWTBL                 6878
027656  016516  000006                  MOV     6(R5),(SP)                      ; *(PTBL.PTR),BNK.NO.HWTBL              6879
027662  016501  000010                  MOV     10(R5),R1                       ; *(PTBL.PTR),ARR.NO.HWTBL              6880
027666  016502  000012                  MOV     12(R5),R2                       ; *(PTBL.PTR),RH.BASE.HWTBL             6881
027672  016503  000014                  MOV     14(R5),R3                       ; *(PTBL.PTR),DUT.HWTBL                 6882
027676  016504  000016                  MOV     16(R5),R4                       ; *(PTBL.PTR),OPTION.HWTBL              6883
027702  016505  000020                  MOV     20(R5),R5                       ; *(PTBL.PTR),CORRECT.HWTBL             6884
027706  006005                  4$:     ROR     R5                              ; CORRECT.HWTBL                         6892
027710  103435                          BLO     5$
027712  012746  002632'                MOV     #LFS,-(SP)                      ;                                       6895
027716  012746  000001                  MOV     #1,-(SP)
027722  010600                          MOV     SP,R0                           ; SP,*
027724  104414                          TRAP    14
027726  012716  001256'                MOV     #SUPRES.MSG,(SP)                ;                                       6896
027732  012746  002556'                MOV     #ONE.MSG,-(SP)
027736  012746  000002                  MOV     #2,-(SP)
027742  010600                          MOV     SP,R0                           ; SP,*
027744  104414                          TRAP    14
027746  012716  001316'                MOV     #RET.DRS.MSG,(SP)               ;                                       6897
027752  012746  002556'                MOV     #ONE.MSG,-(SP)
027756  012746  000002                  MOV     #2,-(SP)
027762  010600                          MOV     SP,R0                           ; SP,*
027764  104414                          TRAP    14
027766  016700  000000G                 MOV     L$UNIT,R0                       ;                                       6898
027772  104451                          TRAP    51
027774  104444                          TRAP    44
027776  062706  000014                  ADD     #14,SP                          ;                                       6894
030002  000537                          BR      8$                              ;                                       6892
030004  005067  040402'          5$:     CLR     ARR$BNK.SEL                     ;                                       6906
030010  010267  040374'                  MOV     R2,ML.ADDR                      ; RH.BASE.HWTBL,*                       6907
030014  010367  040400'                  MOV     R3,ML.DUT                       ; DUT.HWTBL,*                           6908
030020  005304                          DEC     R4                              ; OPTION.HWTBL                          6913
030022  001026                          BNE     6$
```

```
030024  012767  000200  040372'                MOV     #200,COL.BASE                                                   6916
030032  012767  000200  040406'                MOV     #200,INC.BNK                                                    6917
030040  012767  001000  040410'                MOV     #1000,INC.ARR                                                   6918
030046  012767  000177  040416'                MOV     #177,BNK.NUM.SEC                                                6919
030054  012767  000011  040412'                MOV     #11,ARR.SEL.POS                                                 6920
030062  012767  000007  040414'                MOV     #7,BNK.SEL.POS                                                  6921
030070  012767  000200  040376'                MOV     #200,MAX.CHIP.COL                                               6922
030076  000425                                  BR      7$                                                              6913
030100  012767  000400  040372'        6$:     MOV     #400,COL.BASE                                                   6926
030106  012767  001000  040406'                MOV     #1000,INC.BNK                                                   6927
030114  012767  004000  040410'                MOV     #4000,INC.ARR                                                   6928
030122  012767  000777  040416'                MOV     #777,BNK.NUM.SEC                                                6929
030130  012767  000013  040412'                MOV     #13,ARR.SEL.POS                                                 6930
030136  012767  000011  040414'                MOV     #11,BNK.SEL.POS                                                 6931
030144  012767  000400  040376'                MOV     #400,MAX.CHIP.COL                                               6932
030152  032766  000001  000006        7$:     BIT     #1,6(SP)                ; *,SYS.HWTBL                           6941
030160  001451                                  BEQ     9$
030162  016705  040374'                        MOV     ML.ADDR,R5                                                      6944
030166  016566  000010  000012                MOV     10(R5),12(SP)           ; *,ML.REG
030174  016604  000012                        MOV     12(SP),R4               ; ML.REG,MLREG
030200  016705  040400'                        MOV     ML.DUT,R5
030204  042705  177770                        BIC     #177770,R5
030210  142704  000007                        BICB    #7,R4                   ; *,MLREG
030214  050504                                BIS     R5,R4                   ; *,MLREG
030216  016705  040374'                        MOV     ML.ADDR,R5
030222  010465  000010                        MOV     R4,10(R5)               ; MLREG,*
030226  016705  040374'                        MOV     ML.ADDR,R5                                                      6945
030232  016566  000024  000010                MOV     24(R5),10(SP)           ; *,ML.REG
030240  016605  000010                        MOV     10(SP),R5               ; ML.REG,*
030244  006205                                ASR     R5
030246  006205                                ASR     R5
030250  006205                                ASR     R5
030252  000305                                SWAB    R5
030254  042705  177740                        BIC     #177740,R5
030260  005305                                DEC     R5
030262  010567  040434'                        MOV     R5,LST.TSTED.ARR
030266  012767  000004  040436'                MOV     #4,RAND.PASS                                                    6946
030274  012767  000004  040404'                MOV     #4,TSTED.BNK                                                    6947
030302  000532                          8$:     BR      14$                                                             6941
030304  032766  000001  000004        9$:     BIT     #1,4(SP)                ; *,ARR.HWTBL                           6952
030312  001422                                  BEQ     10$
030314  005067  040434'                        CLR     LST.TSTED.ARR                                                   6955
030320  012767  000010  040436'                MOV     #10,RAND.PASS                                                   6956
030326  012767  000004  040404'                MOV     #4,TSTED.BNK                                                    6957
030334  012746  040402'                        MOV     #ARR$BNK.SEL,-(SP)                                              6958
030340  016746  040412'                        MOV     ARR.SEL.POS,-(SP)
030344  012746  000004                        MOV     #4,-(SP)
030350  010146                                MOV     R1,-(SP)                ; ARR.NO.HWTBL,*
030352  004767  000000G                       JSR     PC,BL$PU2
030356  000502                                BR      13$                                                             6952
030360  032766  000001  000002       10$:     BIT     #1,2(SP)                ; *,BNK.HWTBL                           6963
030366  001434                                  BEQ     11$
```

```
030370  005067  040434'                        CLR     LST.TSTED.ARR                   ;                               6966
030374  012767  000015  040436'                MOV     #15,RAND.PASS                   ;                               6967
030402  012767  000001  040404'                MOV     #1,TSTED.BNK                    ;                               6968
030410  012746  040402'                        MOV     #ARR$BNK.SEL,-(SP)              ;                               6969
030414  016746  040412'                        MOV     ARR.SEL.POS,-(SP)
030420  012746  000004                        MOV     #4,-(SP)
030424  010146                                MOV     R1,-(SP)                        ; ARR.NO.HWTBL,*
030426  004767  000000G                       JSR     PC,BL$PU2
030432  012716  040402'                        MOV     #ARR$BNK.SEL,(SP)               ;                               6971
030436  016746  040414'                        MOV     BNK.SEL.POS,-(SP)
030442  012746  000002                        MOV     #2,-(SP)
030446  016646  000014                        MOV     14(SP),-(SP)                    ; BNK.NO.HWTBL,*
030452  004767  000000G                       JSR     PC,BL$PU2
030456  000440                                BR      12$                             ;                               6963
030460  104454                        11$:    TRAP    54                              ;                               6982
030462  000002                                .WORD   2
030464  001070'                                .WORD   HQ.ERR.MSG
030466  000000                                .WORD   0
030470  012746  001144'                        MOV     #HQ.MSG,-(SP)                   ;                               6983
030474  012746  002556'                        MOV     #ONE.MSG,-(SP)
030500  012746  000002                        MOV     #2,-(SP)
030504  010600                                MOV     SP,R0                           ; SP,*
030506  104414                                TRAP    14
030510  012716  001256'                        MOV     #SUPRES.MSG,(SP)                ;                               6984
030514  012746  002556'                        MOV     #ONE.MSG,-(SP)
030520  012746  000002                        MOV     #2,-(SP)
030524  010600                                MOV     SP,R0                           ; SP,*
030526  104414                                TRAP    14
030530  012716  001316'                        MOV     #RET.DRS.MSG,(SP)               ;                               6985
030534  012746  002556'                        MOV     #ONE.MSG,-(SP)
030540  012746  000002                        MOV     #2,-(SP)
030544  010600                                MOV     SP,R0                           ; SP,*
030546  104414                                TRAP    14
030550  016700  000000G                       MOV     L$UNIT,R0                       ;                               6986
030554  104451                                TRAP    51
030556  104444                                TRAP    44
030560  062706  000006                12$:    ADD     #6,SP                           ;                               6961
030564  062706  000010                13$:    ADD     #10,SP                          ;                               6950
030570  005067  000000'                14$:    CLR     PM.COUNT                        ;                               6998
030574  005004                                CLR     R4                              ; INDEX                         7015
030576  010405                        15$:    MOV     R4,R5                           ; INDEX,*                       7017
030600  006305                                ASL     R5
030602  006305                                ASL     R5
030604  005065  000002'                        CLR     PM.LOG(R5)
030610  005065  000004'                        CLR     PM.LOG+2(R5)                    ;                               7018
030614  005204                                INC     R4                              ; INDEX                         7015
030616  020427  000177                        CMP     R4,#177                         ; INDEX,*
030622  003765                                BLE     15$
030624  062706  000014                        ADD     #14,SP                          ;                               6806
030630  000207                                RTS     PC

; Routine Size:  339 words
```

```
; Maximum stack depth per invocation:  19 words



                                        .SBTTL  L$INIT INITIALIZE CODE SECTION
030632  004767  176526          L$INIT::JSR     PC,LINIT                        ;                       7019
030636  104411                          TRAP    11
030640  000207                          RTS     PC

; Routine Size:  4 words
; Maximum stack depth per invocation:  0 words
```

```
;       7022  %sbttl 'MAIN PROM MAINT CONTROL CODE'
;       7023  !
;       7024  ! Main prom maint control code
;       7025  !
;       7026  BGNTST;
;       7027
;       7028  !++
;       7029  ! Functional Description:
;       7030  !       The ML11 memory system with its mostly array technology has the facil-
;       7031  !       ity to offset around known bad memory locations in its memory arrays.
;       7032  !
;       7033  !       Initially, these memory arrays are tested for bad row and  column  ad-
;       7034  !       dress  locations  and the specific offsetting information is stored in
;       7035  !       prom on the array module.  This testing is done on a special 2224 mem-
;       7036  !       ory tester by Memory Manufacturing.
;       7037  !
;       7038  !       The design of the  ML11  has  also  provided  logic  that  when  under
;       7039  !       software  control will update an array modules offsetting information.
;       7040  !       This logic is to be utilized when additional memory cells  are  disco-
;       7041  !       vered bad after the system has left the manufacturing facility.
;       7042  !
;       7043  !    Program Execution Overview
;       7044  !
;       7045  !           1.  Via software question section prompt the operator for
;       7046  !               program parameters and build the run time parameters which
;       7047  !               controls the programs execution.
;       7048  !
;       7049  !           2.  While in ECC-DIS mode do Mass Bus writes,  write  checks  and
;       7050  !               Reads  to  find  all  faulty data chips (bits 0 to 35) in the
;       7051  !               bank under test.
;       7052  !
;       7053  !               Store failing Chip numbers and failing patterns into the bad
;       7054  !               chip table called 'chip_tbl'.
;       7055  !
;       7056  !           3.  If a UNC Err occurred during Step 2
;       7057  !                   then
;       7058  !                       While in data Diagnostic Mode
;       7059  !                       do Single Step DMA Writes and
;       7060  !                       reads to find all faulty CRC Chips
;       7061  !                       (bits 36 to 38) in the
;       7062  !                       bank under test.
;       7063  !                       Store failing chip numbers and
;       7064  !                       failing patterns into 'chip_tbl'.
;       7065  !                   Else
;       7066  !                       Do not perform this step.
;       7067  !
;       7068  !           4.  While in Data Diagnostic Mode do single step DMA  Writes  and
;       7069  !               Load  a  failing  chip  in the bank under test with a failing
;       7070  !               pattern.
;       7071  !
;       7072  !           5.  While in Data Diagnostic Mode do Single Step  DMA  Reads  and
;       7073  !               read all words in all the sectors of the just loaded chip.
```

```
7074  !
7075  !                On each read compare the read chip data to  the  forced  chip
7076  !                data.
7077  !
7078  !                On a compare error load the failing Row address into the error
7079  !                map at the failing Sector number.
7080  !
7081  !            6.  Repeat steps 4 and 5 with all failing patterns for this chip.
7082  !
7083  !            7.  Interigate the error map for this chip and load the blast table
7084  !                at  this  chips nibble and bank position with failing Row and
7085  !                Column offset information.
7086  !
7087  !            8. Zero the error map and repeat steps 4 thru 7 for all  failing
7088  !                chips in this Bank under test.
7089  !
7090  !            9. Zero the error map and chip_tbl and repeat steps 2 thru
7091  !                9 for all remaining banks under test within this  array  mo-
7092  !                dule.
7093  !
7094  !            10. Interigate the blast table at each bank under test  for  this
7095  !                array  module  and see if the newly discovered nibble row and
7096  !                column offsets plus the old nibble  Row  and  Column  offsets
7097  !                exceed 14 for any one nibble position.
7098  !
7099  !                If any bank under test within this array module  nibble  off-
7100  !                sets exceed 14
7101  !                     then
7102  !                        Mask as many errors as possible
7103  !                        (maximum of 14) and leave the
7104  !                        remaining errors unmasked.
7105  !
7106  !                        Report to the operator a condition 'C'
7107  !                        message and continue program execution.
7108  !
7109  !                   Else
7110  !                        continue program execution.
7111  !
7112  !            11. At each bank under test within this array module and at newly
7113  !                discovered nibble Row and Column offset position only, calcu-
7114  !                late the new check sum values.  Store the new check  sum  va-
7115  !                lues  into  the blast table at their respective Row or Column
7116  !                position.
7117  !
7118  !            12. At each bank under test for this array module  and  at  their
7119  !                newly discovered nibble Row and Column offset positions only,
7120  !                blast the module under test UV proms with the new offset  in-
7121  !                formation.
7122  !
7123  !            13. Read the module under test UV  proms  back  and  verify  that
7124  !                blasting was successful.
7125  !
```

```
!                       If blasting was successful
!                           then
!                              Continue program execution
!
!                           else
!                              Report the unsuccessful Prom blast.
!
!                              Report a condition 'D' message and abort
!                              further program execution and return
!                              to DS>
!
!                   14. Verify the blasting has masked the newly discovered errors by
!                       repeating steps 2 and 3 at all tested banks within this array
!                       module.
!
!                       At each iteration the table storing faulty chips and patterns
!                       should be zero.
!
!                       If not zero
!                           then
!                              If this is a degraded mode
!                              Array then
!                              1.  Ignore single bit errors
!
!                              2.  Call Array out for
!                                  replacement of UNC errors
!                                  occur
!
!                           If this is a non degrade
!                              mode Array then
!                              1.  Report single bit errors.
!                                  suspect HW errors
!
!                              2.  Report UNC errors
!                                  suspect HW errors
!
!                   15. Increment the Array Selection count and repeat steps  2  thru
!                       steps 14 until all present Array Modules have been tested.
!
!                   16. Report to operator that Array Maintenance is completed.
!
!                   17. Abort program execution and return to DS>.
!
! Formal Parameters:
!       none
!
! Implicit Inputs:
!       DEGRADE_MOD_REG, BAD_BNK, SBE_LOG, WRT_BUF
!
! Implicit Outputs:
!       none
!
```

```
;          7178  ! Completion Codes:
;          7179  !       none
;          7180  !
;          7181  ! Side Effects:
;          7182  !       none
;          7183  !--
;          7184
;          7185  local
;          7186      TEMP_TSTED_BNK,                          !Temporary storage for the number of tested banks per array
;          7187      TEMP_ARR$BNK_SEL;                        !Temporary storage for the array & bank select variable
;          7188
;          7189  !
;          7190  ! Prompt the operator that the
;          7191  ! prom maintanence program has begun
;          7192  ! and which drive is selected.
;          7193  !
;          7194  PRINTB (ONE_MSG, BGN_MSG);
;          7195  PRINTB (DRV_SEL_PRINT, .ML_DUT);
;          7196  !
;          7197  ! This outer loop will pm all selected
;          7198  ! arrays. It will pm from either one array to the selected
;          7199  ! drives max arrays present.
;          7200  !
;          7201
;          7202  incr ARR_SEL from 0 to .LST_TSTED_ARR do       !Test all selected arrays
;          7203      begin
;          7204      BREAK;
;          7205      DEGRADE_MOD_REG = ZEROES;                  !Clear the degrade mode register
;          7206      BAD_BNK_REG = ZEROES;                      !Clear the bad bank registers flags
;          7207      FLG_REG = ZEROES;                          !Clear the flag register
;          7208      I_BLAST_TBL ();                            !Inti the blast table
;          7209      TEMP_TSTED_BNK = .TSTED_BNK;               !Load number of tested banks into temp storage
;          7210      TEMP_ARR$BNK_SEL = .ARR$BNK_SEL;           !Load the array & bank sel into temp storage
;          7211      !
;          7212      ! This inner loop will find new failing rows &
;          7213      ! columns within bad chips in the banks of the selected
;          7214      ! arrays.  Either 4 banks, if pm'ing sys/array
;          7215      ! or 1 bank if pm'ing a single bank will be tested.
;          7216      !
;          7217      !
;          7218      ! Prompt the operator which array is presently selected
;          7219      !
;          7220      PRINTB (ARR_SEL_PRINT, .ARR$BNK_SEL<.ARR_SEL_POS, ARR$SEL_SIZE>);
;          7221
;          7222      do                                         !Prom maint all the failing chips in this bank
;          7223          begin
;          7224          PRINTB (BNK_SEL_PRINT, .TEMP_ARR$BNK_SEL<.BNK_SEL_POS, BNK$SEL_SIZE>);  !Print
;          7225          FB_CHIPS (.TEMP_ARR$BNK_SEL);          !Find the bad chips in this bank
;          7226          PM_THIS_BANK (.TEMP_ARR$BNK_SEL);      !Prom maint the bad chips in this bank
;          7227          TEMP_ARR$BNK_SEL = .TEMP_ARR$BNK_SEL + .INC_BNK;       !Increment to the next bank
;          7228          TEMP_TSTED_BNK = .TEMP_TSTED_BNK - 1;  !Decrement the tested bank count
;          7229          end
```

```
;        7230      until (.TEMP_TSTED_BNK eql ZERO) or (.FLG_REG [F_ABORT_ARRAY]);     !Repeat until  all banks are tested
;        7231
;        7232      !
;        7233      ! The blast table has now been loaded with all
;        7234      ! newly failing rows and columns within this tested array.
;        7235      !
;        7236
;        7237      if .FLG_REG [F_ABORT_ARRAY]                   !Was the abort array flag set during prom maint
;        7238      then
;        7239          PRINTB (ONE_MSG, ABORT_MSG)               !Report that prom maint for this array is aborted
;        7240      else
;        7241          begin
;        7242          IN_BLAST_TBL (.ARR$BNK_SEL, .TSTED_BNK);        !Interigate the blast table
;        7243
;        7244          if .BAD_BNK_REG eql ZERO                  !Are there any additional errors to blast
;        7245          then
;        7246              PRINTB (ONE_MSG, NO_AD_MSG)           !Print no additional errors message
;        7247          else
;        7248              begin
;        7249              CAL_CHK_SUM ();                       !Calculate the new prom data check sums
;        7250              BL_PROMS (.ARR$BNK_SEL);              !Blast the proms with the newly failing rows/cols
;        7251
;        7252              if .FLG_REG [F_D_CLK_TIME_OUT] then exitloop;
;        7253
;        7254              VER_BLAST (.ARR$BNK_SEL);             !Verify that the blast was successful
;        7255              VER_ERROR_MASK (.ARR$BNK_SEL, .TSTED_BNK);
;        7256                                                    !Verify that the blast masked out the new failing rows/cols
;        7257              end;
;        7258
;        7259          end;
;        7260
;        7261      ARR$BNK_SEL = .ARR$BNK_SEL + .INC_ARR;        !Increment to the next array
;        7262      end;
;        7263
;        7264  !
;        7265  ! Prompt the operator that the
;        7266  ! program execution has ended
;        7267  !
;        7268  PRINTB (ONE_MSG, END_MSG);                       !Report that the program is completed
;        7269  PRINTB (ONE_MSG, RET_DRS_MSG);                   !Print returning to drs> message
;        7270  DODU (.L$UNIT);                                  !Drop this unit
;        7271  DOCLN;                                           !Go to the clear up code section
;        7272  ENDTST;

                                        .SBTTL  $T1 MAIN PROM MAINT CONTROL CODE
030642  004167  000000G          $T1:   JSR     R1,$SAVE4                       ;                                        7021
030646  012746  001000'                 MOV     #BGN.MSG,-(SP)                  ;                                        7194
030652  012746  002556'                 MOV     #ONE.MSG,-(SP)
030656  012746  000002                  MOV     #2,-(SP)
030662  010600                         MOV     SP,R0                           ; SP,*
030664  104414                         TRAP    14
```

```
030666  016716  040400'                MOV     ML.DUT,(SP)                     ;                                       7195
030672  012746  002670'                MOV     #DRV.SEL.PRINT,-(SP)
030676  012746  000002                 MOV     #2,-(SP)
030702  010600                         MOV     SP,R0                           ; SP,*
030704  104414                         TRAP    14
030706  016704  040434'                MOV     LST.TSTED.ARR,R4                ;                                       7202
030712  005001                         CLR     R1                              ; ARR.SEL
030714  000572                         BR      9$
030716  104422                 1$:     TRAP    22                              ;                                       7203
030720  005067  040424'                CLR     DEGRADE.MOD.REG                 ;                                       7205
030724  005067  040420'                CLR     BAD.BNK.REG                     ;                                       7206
030730  005067  040422'                CLR     FLG.REG                         ;                                       7207
030734  004767  154336                 JSR     PC,I.BLAST.TBL                  ;                                       7208
030740  016703  040404'                MOV     TSTED.BNK,R3                    ; *,TEMP.TSTED.BNK                      7209
030744  016702  040402'                MOV     ARR$BNK.SEL,R2                  ; *,TEMP.ARR$BNK.SE                     7210
030750  012746  040402'                MOV     #ARR$BNK.SEL,-(SP)              ;                                       7220
030754  016746  040412'                MOV     ARR.SEL.POS,-(SP)
030760  012746  000004                 MOV     #4,-(SP)
030764  005046                         CLR     -(SP)
030766  004767  000000G                JSR     PC,BL$GT2
030772  010016                         MOV     R0,(SP)
030774  012746  002722'                MOV     #ARR.SEL.PRINT,-(SP)
031000  012746  000002                 MOV     #2,-(SP)
031004  010600                         MOV     SP,R0                           ; SP,*
031006  104414                         TRAP    14
031010  010246                 2$:     MOV     R2,-(SP)                        ; TEMP.ARR$BNK.SE,*                     7224
031012  016746  040414'                MOV     BNK.SEL.POS,-(SP)
031016  012746  000002                 MOV     #2,-(SP)
031022  005046                         CLR     -(SP)
031024  004767  000000G                JSR     PC,BL$GT1
031030  010016                         MOV     R0,(SP)
031032  012746  002770'                MOV     #BNK.SEL.PRINT,-(SP)
031036  012746  000002                 MOV     #2,-(SP)
031042  010600                         MOV     SP,R0                           ; SP,*
031044  104414                         TRAP    14
031046  010216                         MOV     R2,(SP)                         ; TEMP.ARR$BNK.SE,*                     7225
031050  004767  172320                 JSR     PC,FB.CHIPS
031054  010216                         MOV     R2,(SP)                         ; TEMP.ARR$BNK.SE,*                     7226
031056  004767  172362                 JSR     PC,PM.THIS.BANK
031062  066702  040406'                ADD     INC.BNK,R2                      ; *,TEMP.ARR$BNK.SE                     7227
031066  005303                         DEC     R3                              ; TEMP.TSTED.BNK                        7228
031070  062706  000014                 ADD     #14,SP                          ;                                       7223
031074  005703                         TST     R3                              ; TEMP.TSTED.BNK                        7230
031076  001404                         BEQ     3$
031100  032767  000100  040422'        BIT     #100,FLG.REG
031106  001740                         BEQ     2$
031110  032767  000100  040422'  3$:   BIT     #100,FLG.REG                    ;                                       7237
031116  001411                         BEQ     4$
031120  012746  001714'                MOV     #ABORT.MSG,-(SP)                ;                                       7239
031124  012746  002556'                MOV     #ONE.MSG,-(SP)
031130  012746  000002                 MOV     #2,-(SP)
031134  010600                         MOV     SP,R0                           ; SP,*
```

```
031136  104414                          TRAP    14
031140  000452                          BR      8$                      ;                       7237
031142  016746  040402'          4$:    MOV     ARR$BNK.SEL,-(SP)       ;                       7242
031146  016746  040404'                 MOV     TSTED.BNK,-(SP)
031152  004767  164110                  JSR     PC,IN.BLAST.TBL
031156  005767  040420'                 TST     BAD.BNK.REG             ;                       7244
031162  001012                          BNE     5$
031164  012746  001764'                 MOV     #NO.AD.MSG,-(SP)        ;                       7246
031170  012746  002556'                 MOV     #ONE.MSG,-(SP)
031174  012746  000002                  MOV     #2,-(SP)
031200  010600                          MOV     SP,R0                   ; SP,*
031202  104414                          TRAP    14
031204  005726                          TST     (SP)+
031206  000426                          BR      7$                      ;                       7244
031210  004767  172562          5$:    JSR     PC,CAL.CHK.SUM          ;                       7249
031214  016746  040402'                 MOV     ARR$BNK.SEL,-(SP)       ;                       7250
031220  004767  173412                  JSR     PC,BL.PROMS
031224  105767  040422'                 TSTB    FLG.REG                 ;                       7252
031230  100003                          BPL     6$
031232  062706  000022                  ADD     #22,SP
031236  000423                          BR      10$
031240  016716  040402'          6$:    MOV     ARR$BNK.SEL,(SP)        ;                       7254
031244  004767  174574                  JSR     PC,VER.BLAST
031250  016716  040402'                 MOV     ARR$BNK.SEL,(SP)        ;                       7255
031254  016746  040404'                 MOV     TSTED.BNK,-(SP)
031260  004767  175540                  JSR     PC,VER.ERROR.MASK
031264  005726                  7$:     TST     (SP)+                   ;                       7241
031266  066767  040410' 040402'  8$:    ADD     INC.ARR,ARR$BNK.SEL     ;                       7261
031274  062706  000022                  ADD     #22,SP                  ;                       7203
031300  005201                          INC     R1                      ; ARR.SEL               7202
031302  020104                  9$:     CMP     R1,R4                   ; ARR.SEL,*
031304  003604                          BLE     1$
031306  012716  001222'          10$:    MOV     #END.MSG,(SP)           ;                       7268
031312  012746  002556'                 MOV     #ONE.MSG,-(SP)
031316  012746  000002                  MOV     #2,-(SP)
031322  010600                          MOV     SP,R0                   ; SP,*
031324  104414                          TRAP    14
031326  012716  001316'                 MOV     #RET.DRS.MSG,(SP)       ;                       7269
031332  012746  002556'                 MOV     #ONE.MSG,-(SP)
031336  012746  000002                  MOV     #2,-(SP)
031342  010600                          MOV     SP,R0                   ; SP,*
031344  104414                          TRAP    14
031346  016700  000000G                 MOV     LSUNIT,R0               ;                       7270
031352  104451                          TRAP    51
031354  104444                          TRAP    44
031356  062706  000022                  ADD     #22,SP                  ;                       7021
031362  000207                          RTS     PC

; Routine Size:  169 words
; Maximum stack depth per invocation:  22 words
```

```
                                                .SBTTL  T1 MAIN PROM MAINT CONTROL CODE
031364  004767  177252                    T1::
031364                                      1$:     JSR     PC,$T1                          ;                       7271
031370  104466                                      TRAP    66
031372  006000                                      ROR     R0
031374  103773                                      BLO     1$
031376  000207                                      RTS     PC

; Routine Size:  6 words
; Maximum stack depth per invocation:  0 words
```

```
;          7273  %sbttl 'CLEAN UP CODE SECTION'
;          7274  !
;          7275  ! Clean up code section
;          7276  !
;          7277  BGNCLN;
;          7278  CLR_MBUS;
;          7279  ENDCLN;


                                        .SBTTL  LCLEAN CLEAN UP CODE SECTION
031400  010146                  LCLEAN: MOV     R1,-(SP)                ;                                       7272
031402  016700  040374'                MOV     ML.ADDR,R0              ;                                       7277
031406  016046  000010                MOV     10(R0),-(SP)            ; *,ML.REG
031412  011601                        MOV     (SP),R1                 ; ML.REG,MLREG
031414  152701  000040                BISB    #40,R1                  ; *,MLREG
031420  016700  040374'                MOV     ML.ADDR,R0
031424  010160  000010                MOV     R1,10(R0)               ; MLREG,*
031430  016700  040374'                MOV     ML.ADDR,R0
031434  016046  000010                MOV     10(R0),-(SP)            ; *,ML.REG
031440  012601                        MOV     (SP)+,R1                ; ML.REG,MLREG
031442  016700  040400'                MOV     ML.DUT,R0
031446  042700  177770                BIC     #177770,R0
031452  142701  000007                BICB    #7,R1                   ; *,MLREG
031456  050001                        BIS     R0,R1                   ; *,MLREG
031460  016700  040374'                MOV     ML.ADDR,R0
031464  010160  000010                MOV     R1,10(R0)               ; MLREG,*
031470  005726                        TST     (SP)+                   ;                                       7272
031472  012601                        MOV     (SP)+,R1
031474  000207                        RTS     PC

; Routine Size:  31 words
; Maximum stack depth per invocation:  4 words


                                        .SBTTL  L$CLEAN CLEAN UP CODE SECTION
031476  004767  177676          L$CLEAN::
                                        JSR     PC,LCLEAN               ;                                       7278
031502  104412                        TRAP    12
031504  000207                        RTS     PC

; Routine Size:  4 words
; Maximum stack depth per invocation:  0 words

;          7280  end
;          7281
;          7282  eludom

                                        ; OTS external references
```

```
                        .GLOBL  BL$GT2, $SAVE5, $SAVE4, $SAVE3
                        .GLOBL  $SAVE2, BL$PU2, BL$GT1, BL$DIV
                        .GLOBL  BL$MOD




; Size:         6563 code + 9592 data words
; Run Time:     01:20.5
; Elapsed Time: 03:07.8
; Memory Used:  133 pages
; Compilation Complete
```

PSECT Storage Map

Name	Loc	Length	/Start	Length	File	Module	Ident
	002000	000000					
$CODE$	002000	033016					
			002000	000200	PMSKL2	BSKEL2	REV B
			002200	000312	PMSKL3	BSKEL3	REV B
			002512	000046	B16ROT	B16ROT	2.6
			002560	000106	B16PG2	B16PG2	2.3
			002666	000104	B16PG4	B16PG4	2.4
			002772	000316	B16MUL	B16MUL	2.7
			003310	031506	PMSKL4	BSKEL4	REV B
OWN	035016	040440					
$PLIT$	075456	005450					
			075456	000530	PMSKL3	BSKEL3	REV B
			076206	004720	PMSKL4	BSKEL4	REV B
XXX	103126	000324					
			103126	000076	RANDOM	RANDOM	
			103224	000106	B16SAV	B16SAV	2.3
			103332	000120	B16PN1	B16PN1	
XYZ	103452	000010					
			103452	000010	LSTAD	LSTAD	NONE
. ABS.	103462	000000					

Map of Global Symbols

SEQ 0303

Name	Value	PSECT	Module	ABS Value	
$SAVE2	000000	XXX	B16SAV	103224	$CODE$
$SAVE3	000014	XXX	B16SAV	103240	$CODE$
$SAVE4	000032	XXX	B16SAV	103256	$CODE$
$SAVE5	000052	XXX	B16SAV	103276	$CODE$
BL$DIV	000224	$CODE$	B16MUL	003216	$CODE$
BL$GT1	000000	XXX	B16PN1	103332	$CODE$
BL$GT2	000000	$CODE$	B16PG2	002560	$CODE$
BL$MOD	000236	$CODE$	B16MUL	003230	$CODE$
BL$PU2	000000	$CODE$	B16PG4	002666	$CODE$
BL$SHF	000250	$CODE$	B16MUL	003242	XXX
L$AU	000302	$CODE$	BSKEL3	002502	$CODE$
L$AUTO	000256	$CODE$	BSKEL3	002456	$CODE$
L$CLEA	031476	$CODE$	BSKEL4	035006	$CODE$
L$DESC	000040	$CODE$	BSKEL3	002240	$CODE$
L$DLY	000116	$CODE$	BSKEL2	002116	$CODE$
L$DU	000270	$CODE$	BSKEL3	002470	$CODE$
L$DVTY	000000	$CODE$	BSKEL3	002200	$CODE$
L$HARD	000102	$CODE$	BSKEL3	002302	$CODE$
L$INIT	030632	$CODE$	BSKEL4	034142	$CODE$
L$LAST	000004	XYZ	LSTAD	103456	$CODE$
L$RPT	000244	$CODE$	BSKEL3	002444	$CODE$
L$SOFT	000226	$CODE$	BSKEL3	002426	$CODE$
L$UNIT	000012	$CODE$	BSKEL2	002012	$CODE$
RANDAT	000074	XXX	RANDOM	103222	$CODE$
RANGEN	000000	XXX	RANDOM	103126	$CODE$
SEED1	000066	XXX	RANDOM	103214	$CODE$
SEED2	000070	XXX	RANDOM	103216	$CODE$
SEED3	000072	XXX	RANDOM	103220	$CODE$
SUMMAR	000570	$CODE$	BSKEL4	004100	$CODE$
T$PTHV	000000		LSTAD	000000	$CODE$
T1	031364	$CODE$	BSKEL4	034674	$CODE$