Box 38 RAMP + error recovery
folder 14

8 of 11 102739899

5.0 A note on the implementation

The console cannot determine the exact amount of error-free memory that BOOT requires. This number may be significantly less than 256K for a particular configuration.

If the console were designed to refuse to load BOOT when every 256K section had an error, it might make the system unavailable unnecessarily. A better choice would be to configure such that the error was in the highest possible memory location and load BOOT. If BOOT halted, the console could then declare the machine unusable.

+---------------+
! d i g i t a l ! I N T E R O F F I C E M E M O R A N D U M
+---------------+

TO: Ted Hess
Pat Sullivan
Ron McLean
Jim Jones
John Kirchoff
TOPS-20 Group
TOPS-10 Group

CC: Peter Hurley
Don Lewine

DATE: September 22, 1980

FROM: Judy Hall

DEPT: DEC-20 S. E. Dept.

LOC: MR 1-2/E37

DTN: 231-6421

SUBJ: Memory Errors at BOOT Time

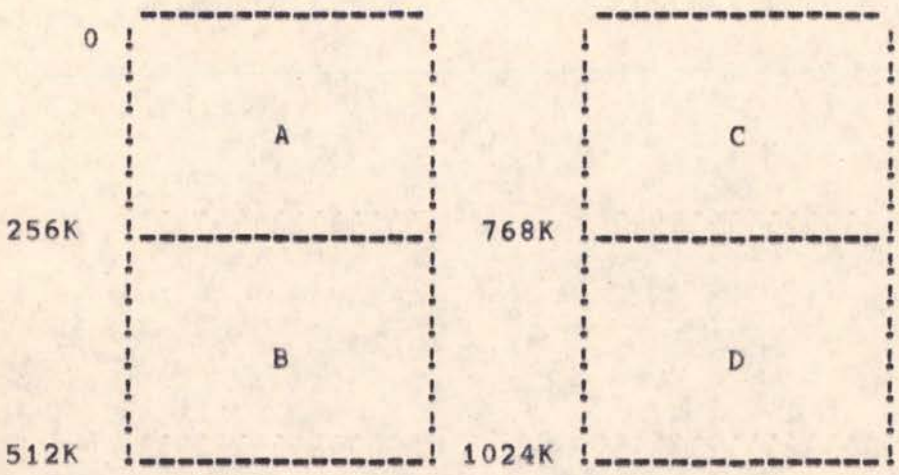
This memo describes my understanding of how we will deal with double-bit errors when TOPS-20 is being loaded on the Jupiter. My purpose in writing it is to uncover differences of opinion among the groups that are concerned with this issue. Therefore, if you disagree with my facts or my conclusions, please let me know.

1.0 Memory on Jupiter

Each memory module on the Jupiter will be 512K words in size. By changing the interleaving, the console will be able to configure out half of each module. Of course, it can also configure out the entire module. No finer reconfiguration will be possible.

The figure below illustrates the possibilities. (In the interest of clarity, I have used the term "section" to describe half of a module.) The console can swap sections within a module (A with B, C with D), or it can swap modules (A-B with C-D).

This means that if any of the 4 sections of 256K is error-free, the console can configure that section to start at location 0. Note, however, that if one section in each module is unavailable, no amount of reconfiguration will produce contiguous memory.



2.0 The monitor

BOOT loads the resident monitor into memory starting at page 0. Memory must be contiguous to the end of the resident monitor. The number of pages required varies, depending on the configuration and the amount of resident code, but never exceeds 256K.

Once the monitor is loaded, it can load the swappable monitor into non-contiguous memory.

3.0 The question

The Diagnostics people plan to have the console zero all of the -10's memory and perform a double-bit error scan before loading BOOT at system startup. The console will then pass to BOOT a map describing those pages that have double-bit errors (Engineering Functional Spec, Edition 3, page 16-28).

At a minimum, we will have to change BOOT to pass this map to the resident monitor. However, it would be nice if we could be guaranteed that the memory into which BOOT and the resident monitor would be loaded would be error-free. This would avoid some coding effort and thus eliminate a source of bugs in BOOT.

Whether or not such a guarantee is reasonable depends largely on the expected failure rate of the memory. If we can expect a double-bit error to occur infrequently, and we can expect SPEAR's analysis to be effective in predicting such failures, then it seems reasonable to assume that we can have 256K of error-free memory at system startup.

4.0 The conclusion

I would like to conclude that the monitor will require error-free memory in location 0 through 256K. If you believe this is not a reasonable conclusion, please let me know.

digital

INTEROFFICE MEMORANDUM

TO: Distribution

DATE: October 9, 1980
FROM: Max Weinfuss
DEPT: CSSE/LSG
EXT: 231-6875
LOC/MAIL STOP: MR1-1/S35

SUBJECT: RAMP Features

The attached are the RAMP features we agreed upon at the review on 17 September 1980. We believe we made all the requested modifications. Please review the document. If we have missed anything, or if you have any concerns, please contact me for resolution.

If we receive no requests for modification, we will consider this our base document. Further changes can and will be made via procedures in Chapter 2 of the M.E.G. Project Plan.

MW/mb
Attachment

FIRST REVIEW 17-SEP-80

REVISION 2

6-OCT-80

NOTE

THIS DOCUMENT IS INTENDED TO REFLECT THE RAMP REQUIREMENTS FOR THE JUPITER SYSTEM. IT IS NOT A FUNCTIONAL SPECIFICATION ON HOW THE JUPITER WORKS. IT IS A DETAILED DOCUMENT OF THE FEATURES THAT HAVE BEEN OR ARE TO BE DESIGNED INTO THE JUPITER SYSTEM TO MAKE IT A HIGHLY RELIABILE AND MAINTAINABLE SYSTEM.

APPENDIX A JUPITER HARDWARE RAMP REQUIREMENTS

A.1 CONSOLE RELIABILITY IMPLEMENTATION A-4
A.1.1 CONSOLE BOARD <IMPLEMENTED> A-5
A.1.2 DUAL (RL02) CONSOLE LOAD DEVICES <IMPLEMENTED> . A-6
A.2 MAINTAINABILITY IMPLEMENTATION A-7
A.2.1 CONSOLE BOARD <IMPLEMENTED> A-7
A.2.1.1 ASYNC LINE LOOPBACK <IMPLEMENTED> A-7
A.2.1.2 DUAL RL02 <IMPLEMENTED> A-7
A.2.1.2.1 LOAD DEVICE LOOPBACK <IMPLEMENTED> A-8
A.2.2 CONSOLE BOARD <TO BE IMPLEMENTED> A-8
A.2.2.1 DISABLEING OF RD <TO BE IMPLEMENTED> A-8
A.2.3 CONSOLE BOARD <PROPOSED> A-9
A.3 CPU RELIABILITY IMPLEMENTATION A-10
A.3.1 MVE MODULE (EBOX) <IMPLEMENTED> A-11
A.3.2 SH MODULE (EBOX) <IMPLEMENTED> A-11
A.3.3 ESE AND CRA MODULE (EBOX) <IMPLEMENTED> . . . A-12
A.3.4 UNCHECKED CONTROL LOGIC <TO BE IMPLEMENTED> . A-13
A.3.5 IDL AND IDR MODULES (IBOX) <TO BE IMPLEMENTED> A-13
A.3.6 APA ISOLATION <TO BE IMPLEMENTED> A-14
A.3.7 TAG MODULE (IBOX) <TO BE IMPLEMENTED> A-14
A.3.8 INX MODULE (IBOX) <TO BE IMPLEMENTED> A-15
A.3.8.1 PIPELINE STOP MECHANISM <TO BE IMPLEMENTED> A-15
A.3.8.2 HARDWARE RECOVERY AND RETRY LOGIC <TO BE IMPLEMENTED> A-15
A.3.9 MBOX GENERAL <TO BE IMPLEMENTED> A-16
A.3.10 CACHE MEMORIES (MBOX) <TO BE IMPLEMENTED> . . A-16
A.3.11 MOS MEMORY <TO BE IMPLEMENTED> A-17
A.3.12 SCA MODULE <PROPOSED> A-18
A.3.12.1 REDUNDANT ALU'S <PROPOSED> A-18
A.3.12.2 TIMERS AND METERS <PROPOSED> A-18
A.3.13 INX MODULE <PROPOSED> A-18
A.3.14 APA DISABLE <PROPOSED> A-19
A.3.15 HUNG CONDITIONS <PROPOSED> A-19
A.3.16 SYSTEM CLOCK CONTTROL <PROPOSED> A-19
A.3.16.1 PHASE CHECK A-19
A.3.16.2 BACKPLANE DISTRIBUTION A-20
A.3.17 MBOX ERROR RECOVERY <PROPOSED> A-20
A.3.18 MOS MEMORY ADDRESS PARITY <PROPOSED> A-20
A.3.19 MBOX INTERNAL ADDRESS PARITY <PROPOSED> . . . A-20
A.3.20 CACHE USE BIT TABLE AND REFILL RAM <PROPOSED> A-21
A.3.21 DATA CACHE WRITTEN AND VALID BIT FAILURES <PROPOSED> A-21
A.3.22 PAGER CACHE VALID AND KEEP BIT FAILURES <PROPOSED> A-21
A.3.23 LOSS OF MBOX OR IO CLOCKS <PROPOSED> A-21
A.3.24 BAD IO DATA <PROPOSED> A-21
A.4 CPU MAINTAINABILITY IMPLEMENTATION A-22
A.4.1 OVERALL ERROR CONTROL AND DETECTION LOGIC <IMPLEMENTED> A-23
A.4.2 DIAGNOSTIC INTERFACE <IMPLEMENTED> A-23
A.4.2.1 ACTIVE SCAN PATHS <IMPLEMENTED> A-23
A.4.2.2 PASSIVE SCAN PATH <IMPLEMENTED> A-24
A.4.2.3 SCAN READ AND LOAD <IMPLEMENTED> A-24
A.4.2.4 DIAGNOSTIC SCAN PATH VISIBILITY <IMPLEMENTED> A-24

A.4.3 FAULT ISOLATION BY INTERNAL HARDWARE <TO BE IMPLEMENTED> A-25
A.4.4 MEMORY RECONFIGURATION <TO BE IMPLEMENTED> . . A-28
A.4.5 CONSOLE LOGGING OF MBOX ERRORS <TO BE IMPLEMENTED> A-28
A.4.6 MOS MEMORY MODULE ISOLATION <TO BE IMPLEMENTED> A-28
A.4.7 SYSTEM CLOCK CONTROL <PROPOSED> A-28
A.4.8 CONTROL BY CONSOLE A-29
A.4.8.1 FREQUENCY CONTROL A-29
A.4.8.2 ENABLE DISABLE FEATURE A-29
A.4.8.3 TIC AND PULSE MODES A-29
A.4.9 ERROR PROPAGATION AND MULTIPLE ERRORS <PROPOSED> A-30
A.4.10 ERROR INFO RETRIEVAL ON A "DYING/DEAD" MACHINE <PROPOSED> A-30
A.4.11 PIPELINE OVERRIDE <PROPOSED> A-30
A.4.12 MBOX HARDWARE FRU ISOLATION <PROPOSED> A-31
A.4.13 MBOX ERROR LOGGING <PROPOSED> A-31
A.4.14 PC TRACE MODE <PROPOSED> A-31
A.4.15 ADVANCED ADDRESS BREAK FACILITY <PROPOSED> . . A-31
A.5 IOBOX RELIABILITY IMPLEMENTATION A-33
A.5.1 INTERNAL PARITY <TO BE IMPLEMENTED> A-33
A.5.2 IOBUS ARBITRATION <TO BE IMPLEMENTED> A-33
A.5.3 INCOMPLETE CYCLES <TO BE IMPLEMENTED> A-33
A.5.4 MTBF <TO BE IMPLEMENTED> A-33
A.6 MAINTAINABILITY IMPLEMENTATION A-33
A.6.1 ERROR RECORDING <TO BE IMPLEMENTED> A-33
A.6.2 LOOPBACK <TO BE IMPLEMENTED> A-34
A.7 PORTS A-35
A.8 RELIABILITY IMPLEMENTATION A-35
A.8.1 INTERNAL PARITY <TO BE IMPLEMENTED> A-35
A.8.2 MICRO-CODE ERRORS <TO BE IMPLEMENTED> A-35
A.8.3 EXTERNAL ERRORS <TO BE IMPLEMENTED> A-35
A.8.4 INCOMPLETE CYCLES <TO BE IMPLEMENTED> A-35
A.8.5 DATA INTEGRITY <TO BE IMPLEMENTED> A-35
A.8.6 CLOCK MARGINING <TO BE IMPLEMENTED> A-36
A.8.7 MAINTAINABILITY IMPLEMENTATION A-36
A.8.8 DISABLE FEATURE <TO BE IMPLEMENTED> A-36
A.8.9 LOOPBACK <TO BE IMPLEMENTED> A-36
A.8.10 POWER-UP STATUS CHECK <TO BE IMPLEMENTED> . . A-36
A.8.11 ERROR RECORDING <TO BE IMPLEMENTED> A-37
A.9 ADAPTORS A-38
A.10 RELIABILITY IMPLEMENTATION A-38
A.10.1 INTERNAL PARITY <TO BE IMPLEMENTED> A-38
A.10.2 INCOMPLETE CYCLES <TO BE IMPLEMENTED> A-38
A.11 MAINTAINABILITY IMPLEMENTATION A-38
A.11.1 LOOPBACK <TO BE IMPLEMENTED> A-38
A.11.2 DYNAMIC DISABLE <TO BE IMPLEMENTED> A-38
A.11.3 RAMP PARTITIONING <TO BE IMPLEMENTED> A-39
A.11.4 ERROR RECORDING <TO BE IMPLEMENTED> A-39
A.11.5 ON-LINE MAINTENANCE <PROPOSED> A-39
A.12 MASSBUS ADAPTOR A-40
A.13 RELIABILITY IMPLEMENTATION A-40
A.13.1 INTERNAL PARITY <IMPLEMENTED> A-40
A.13.2 MASSBUS INTEGRITY <IMPLEMENTED> A-40

A.13.3 DATA INTEGRITY <IMPLEMENTED> A-40
A.14 MAINTAINABILITY IMPLEMENTATION A-40
A.14.1 LOOPBACK <IMPLEMENTED> A-40
A.14.1.1 DYNAMIC DISABLE A-40
A.14.2 INCOMING PARITY CHECK <IMPLEMENTED> A-41
A.14.3 ON-LINE MAINTENANCE <PROPOSED> A-41
A.15 PACKET BUFFER A-42
A.16 RELIABILITY IMPLEMENTATION A-42
A.16.1 INTERNAL PARITY <TO BE IMPLEMENTED> A-42
A.17 MAINTAINABILITY IMPLEMENTATION A-42
A.17.1 LOOPBACK <IMPLEMENTED> A-42
A.17.2 DIAGNOSTIC FOCUS <IMPLEMENTED> A-42
A.18 UNIBUS ADAPTOR (TBD) A-43
A.19 BLOCK MULTIPLEXER (TBD) A-43

APPENDIX B JUPITER MAINTENANCE SOFTWARE RAMP REQUIREMENTS

B.1 DIAGNOSTIC/CONSOLE RAMP- RELIABILITY IMPLEMENTATION B-3
B.1.1 CONSOLE RELIABILITY <IMPLEMENTED> B-3
B.1.2 DUAL CONSOLE LOAD DEVICES <TO BE IMPLEMENTED> . B-3
B.1.3 VT100 <TO BE IMPLEMENTED> B-3
B.1.4 RL02 PROTECTION SCHEME <TO BE IMPLEMENTED> . . . B-4
B.1.5 JUPITER DIAGNOSTIC PACKAGE <TO BE IMPLEMENTED> . B-4
B.1.6 INTEGRITY CHECK <TO BE IMPLEMENTED> B-5
B.1.7 ERROR RECOVERY/RECORDING <TO BE IMPLEMENTED> . . B-5
B.1.8 ENVIRONMENTAL MONITORING <TO BE IMPLEMENTED> . . B-6
B.1.9 MEMORY RECONFIGURABILITY <TO BE IMPLEMENTED> . . B-6
B.1.10 PRIVILEGED COMMAND PROTECTION <PROPOSED> B-6
B.1.11 REDUNDANT BOOTSTRAP <PROPOSED> B-7
B.2 DIAG/MAINTENANCE SOFTWARE MAINTAINABILITY IMPLEMENTATION B-8
B.2.1 HARDCORE <IMPLEMENTED> B-8
B.2.2 SCAN PATHS <IMPLEMENTED> B-8
B.2.3 MAP OF NON-VOLATILE RAM <TO BE IMPLEMENTED> . . B-9
B.2.4 ISOLATION DIAGNOSTICS <TO BE IMPLEMENTED> . . B-10
B.2.5 FUNCTIONALS <TO BE IMPLEMENTED> B-11
B.2.5.1 CPU <TO BE IMPLEMENTED> B-11
B.2.5.2 EXERCISER <TO BE IMPLEMENTED> B-11
B.2.5.3 IO B-11
B.2.5.4 ADAPTORS <TO BE IMPLEMENTED> B-12
B.2.5.5 DEVICE DIAGS <TO BE IMPLEMENTED> B-12
B.2.6 RL02 FILE UTILITY <TO BE IMPLEMENTED> B-12
B.2.7 TRACE UTILITY <TO BE IMPLEMENTED> B-12
B.2.8 RD PARTITIONING <TO BE IMPLEMENTED> B-12
B.2.9 KLINIK <TO BE IMPLEMENTED> B-13
B.2.10 DN2X LINK <PROPOSED> B-13
B.2.11 RECOGNITION <PROPOSED> B-13
B.2.12 COURSE OF ACTION BUFFER <PROPOSED> B-14

APPENDIX C JUPITER SOFTWARE RAMP REQUIREMENTS (TBS)

APPENDIX A

JUPITER HARDWARE RAMP REQUIREMENTS

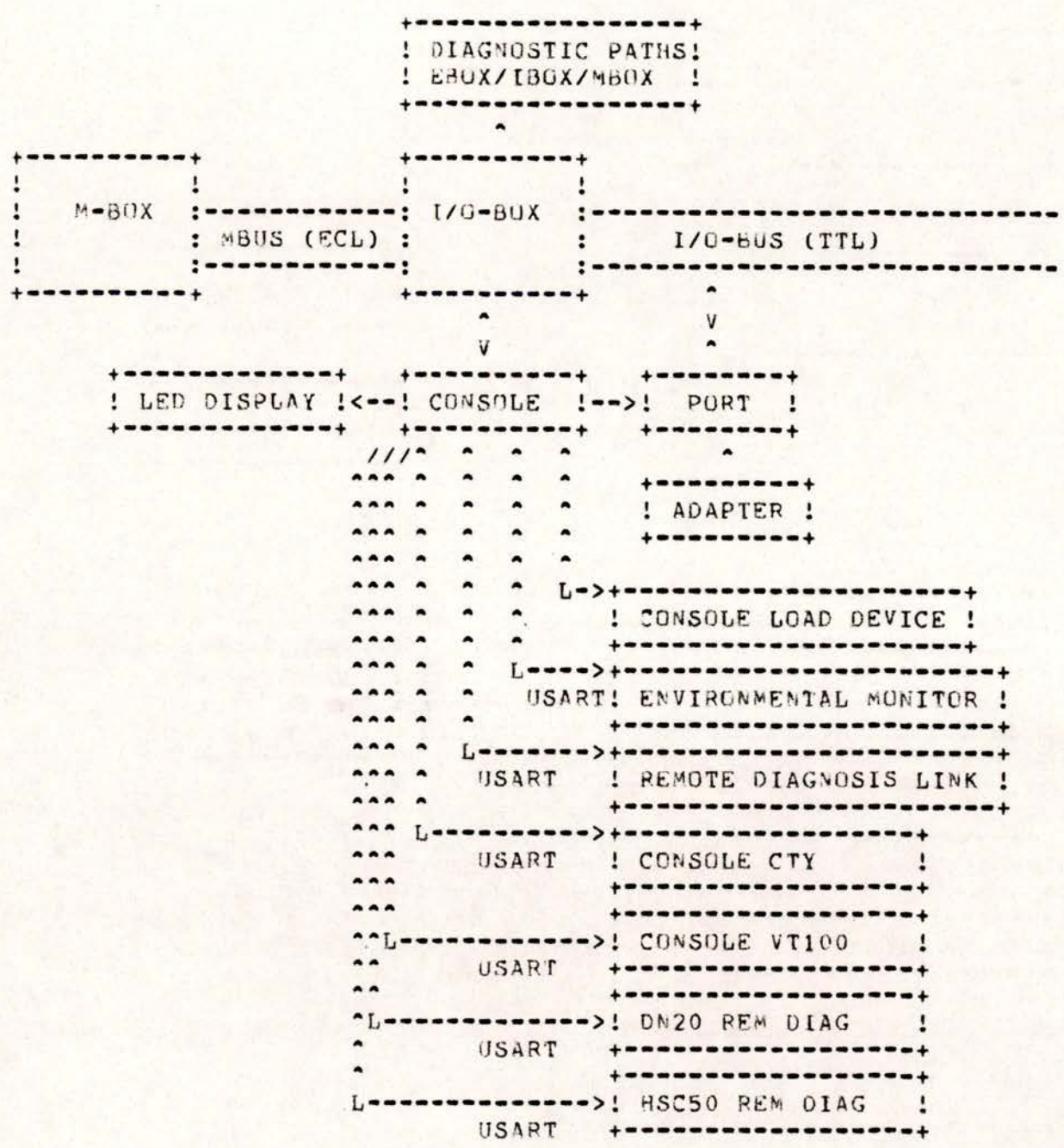
CONSOLE

The RAMP requirements in this section apply to the JUPITER Maintenance Console Subsystem. The JUPITER Maintenance Console Subsystem is the main focal point for implementation of RAMP features being designed in the JUPITER Mainframe (KC10).

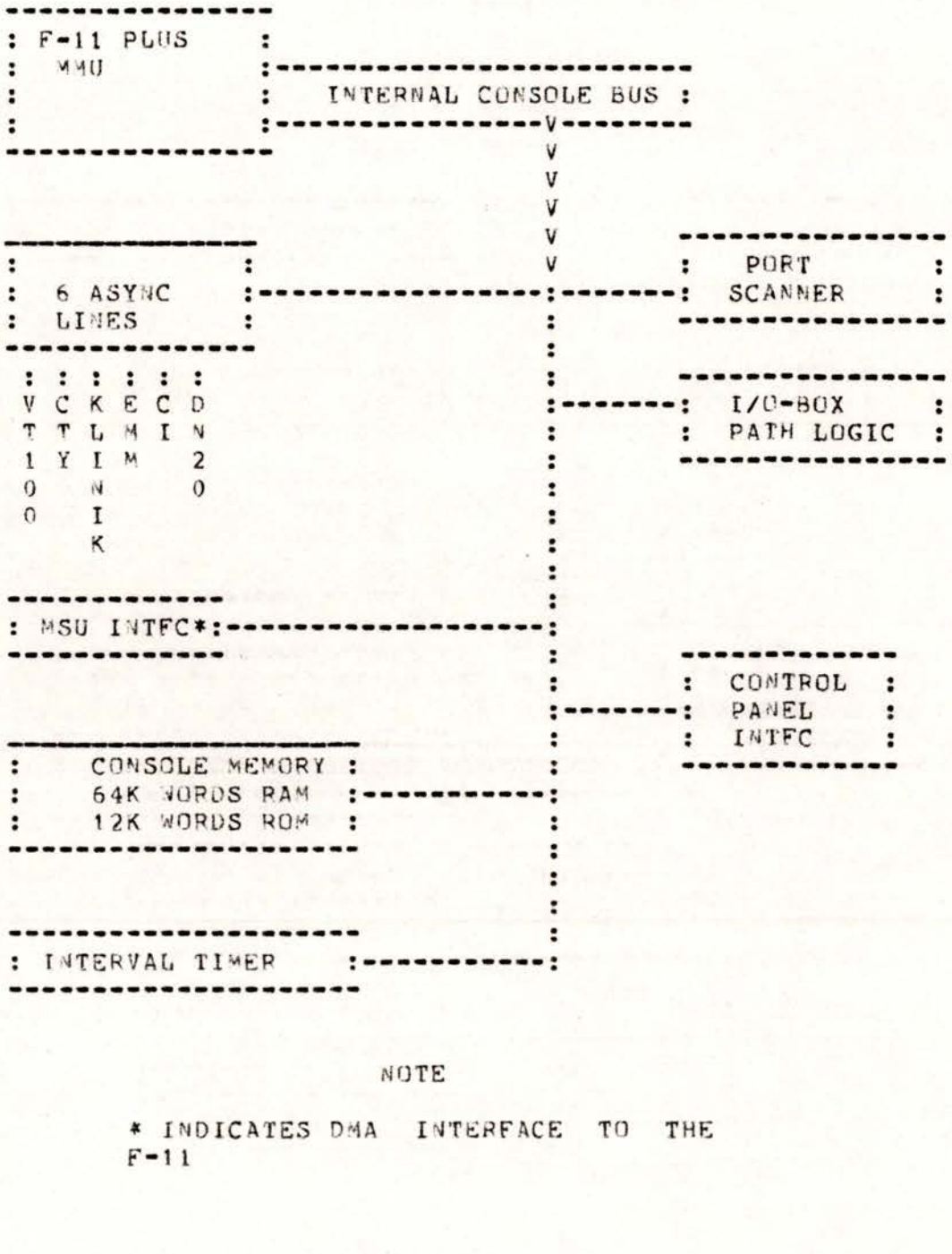
The prime motivation for the existence of the Maintenance Console subsystem (besides operator interface) is to meet the high availability requirements of the JUPITER system. The console and its load device are necessary elements for the operation of the system. The console subsystem itself is designed with high reliability and rapid serviceability as a goal.

The hardware configuration of the JUPITER Maintenance Console consists of a single extended hex FONZ-11 based module, an operator control panel a console load device (and control), an LA34 hardcopy terminal, a VT100 display terminal, a main interface to the I/O BOX including diagnostic interfaces to the KC10 CPU, diagnostic shift path to each of the PORT modules and interfaces connecting to the KLINIK modem, the Environmental Monitoring Modules (EMM) of the mainframe and JUPITER peripherals requiring direct remote diagnosis hookup (i.e. HSC50, MERCURY, DN20).

JUPITER CONSOLE SUBSYSTEM



JUPITER INTERNAL BUS CONNECTIONS



NOTE

* INDICATES DMA INTERFACE TO THE F-11

The console will contribute to the high availability goal of the JUPITER primarily by supporting the fault tolerant design of the system. Extensive error detection and correction in the rest of the cpu cluster will depend upon the console in many instances to become involved in the recovery procedures. Console ramp required to facilitate system continuation via a variety of methods, contributing to reliability of the system relative to

the customers application and scheduled maintainance is discussed here in the RELIABILITY section. Console ramp required to facilitate returning the system to its original condition is addressed in the MAINTAINABILITY section.

A.1 CONSOLE RELIABILITY IMPLEMENTATION

Emphasis has been placed upon the inherent system availability during the hardware selection phase of the console subsystem. The expected result is that the most frequent hardware failures can be bypassed or deferred for scheduled maintenance. The single point of failures resulting in system crashes will come predominately from the console module itself and the RLV-11 control board for the RL02's. While the MTBF analysis for the console subsystem has not been completed at this time, it is reasonable to assume that this hardware will not contribute to system failures more frequently than once per year.

***** IMPLEMENTED *****

A.1.1 CONSOLE BOARD <IMPLEMENTED>

The contents of this board have been carefully selected to meet the reliability goal for the console and yet supply the functionality required to obtain high reliability and optimum serviceability for the JUPITER system. The GOAL used for modelling the selection was greater than 10,000 hours MTBF.

ESSENTIAL HARDWARE CONTENTS

FONZ-11 CHIP SET

SELF CHECKING PROM CODE

6 (2K * 8 BIT) PROMS
16 ON BOARD LEDS

64K WORDS RAM MEMORY

256 WORDS NV RAM
3 (2 X USARTS) ASYNC DIPS

I/O BOX INTERFACE

INTERVAL TIMER

'CLD' BUS - MODIFIED QBUS

BATTERY BACKUP (FOR TOY CLOCK AND NVRAM)

In the design of raw functionality, care has been taken to avoid unnecessary crashes due to non critical failures of the console board. A list of identified non critical failures and corrective actions to be taken follows:

ERROR	CORRECTIVE ACTION
Single bit memory errors -or- or other single bit data bus errors on console board	Soft error detection single bit via classical PDP-11 parity followed by a LRC like bit correction scheme-WHENEVER the Runtime Support Package is resident. (KCRSP)
USART failures	USARTS are initially tested by the console self check code resident in PROM in loopback mode. A detected USART failure

causes the self check code to mark the failing event in non-volatile ram,leave indication in the appropriate 1 of 16 on board LEDs, notification to the operator that the USART is inoperable. (See MAINTAINABILITY IMPLEMENTED section for further enhancement)

NOTE

THAT IN ALL CASES, CORRECTABLE RECOVERABLE OR CIRCUMVENTED HARDWARE ERRORS DETECTED ARE REPORTED TO SPEAR.

To minimize the effect of crashes occurring from power outages,BATTERY BACKUP has been implemented on the console board. The battery backed up portions of the CONSOLE are the TOY (Time Of Year clock) and a 256 WORD RAM memory plus minimal logic involved with storage control of the RAM. Engineering has chosen a clock chip and battery unit from Catalyst Research Corporation. This battery is spec'd at 850ma hours which in consideration of the total calculated current drain of 60 microamps should result in a battery lifetime expectancy of 16 Years of active life. This feature allows for the requirement of power fail recovery of the JUPITER system. For battery maintenance, refer to the hardware maintainability implemented (console board) section. For a layout of the Non-Volitile RAM contents, refer to the Diagnostic reliability/maintainability (to be implemented) sections.

A.1.2 DUAL (RL02) CONSOLE LOAD DEVICES <IMPLEMENTED>

This scheme provides for 20 megabytes of on-line storage which is adequate to ensure that all console system and diagnostic software will be available without the need for human interaction.

The RL interface supports CRC-16 redundency checking.

A.2 MAINTAINABILITY IMPLEMENTATION

The console processor is contained on a single board which is self checked. A power on indicator mounted on the operator's panel indicates whether or not TTL power has been applied to the console board. Thus if the failure mode is such that no response occurrs after the console start switch or the system start switch is depressed, the fault can be quickly isolated to either the TTL power or the single console board.

***** IMPLEMENTED *****

A.2.1 CONSOLE BOARD <IMPLEMENTED>

The console board has a self contained PROM code (KCROM) that will, upon initialization, provide a step by step sequence of console subsystem functionality checks beginning with a check of the F@NZ-11 micro processor and extending to each functional section of the board until it has verified sufficient operational capability to load the console software from the RL02. As each test is completed successfully, 1 of a string of 16 LED's will be lit. In addition, information will be transmitted to the VT100 and LA38 in a format such as:

CONSOLE TESTS OK

WHERE EACH LETTER REPRESENTS AN LED LIT DURING THE TEST SEQUENCING

Any failures which result during the tests that do not clear during a retry will result in test termination if the function is critical. Failures from non-critical functions such as some usarts will result in operator notification and error logging for deferred maintenance.

A.2.1.1 ASYNC LINE LOOPBACK <IMPLEMENTED> -

Loopback capability is provided for console communications lines.

A.2.1.2 DUAL RL02 <IMPLEMENTED> -

The RL system supports parity checking on the main data path. In addition having dual drives enhances maintainability as well as reliability since the the diagnostics can be loaded from either drive.

A.2.1.2.1 LOAD DEVICE LOOPBACK <IMPLEMENTED> -

Loopback testing of console interface logic to the RLV11 will be done up to the point where the modified q-bus leaves the console board.

***** TO BE IMPLEMENTED ******

A.2.2 CONSOLE BOARD <TO BE IMPLEMENTED>

Although the normal service mode for handling a malfunctioning console board will be to replace it with a spare, there is a requirement to socket some of the elements. USARTS shall be socketed to facilitate the ability to change out a single failing USART when the option presents itself.The reason here is that a substantial number of console board failures will come from one of three dual USART chips. USART chips will be spared to the field for such occasions. Also the three PROM chips should be socketed to facilitate the relatively high probability that ECO's will be required to the PROM's due to design deficiency or product enhancement during or after the initial production phase. This will allow a faster revision update on the production floor and possibly lessen the total logistics recycle of console boards in field for said PROM changes.

The battery on the console board will be raplaceable. The self test feature of the console will check for the charged presence of the battery and report the fault when detected. Because of the long expected lifetime, it is not required to change this battery during PM periods.

A.2.2.1 DISABLEING OF RD <TO BE IMPLEMENTED> -

Corporate Field Service requires that Remote Diagnosis be an option on all future computer products. We have discussed this requirement with Diagnostic Engineering and have reached agreement that partitioning of the REMOTE ENTRY support code into one only of the three PROMS would be the easiest and safest way to implement RD as an option. This means that the console PROM code would be designed to operate without RD in two 2K PROMS.

***** PROPOSED *****

A.2.3 CONSOLE BOARD <PROPOSED>

In the case of non critical failures within the Console subsystem, such as a USART failure (hard), the operator should be allowed to reassign the associated device to a working Usart if required. This would be accomplished after the console software is resident and would require something like a REASSIGN command and physical swapping of the cable at the USART connection area. This would result in a mapping of the reassignment in the non-volatile ram so that the proper protocall and baud rate selection is made. This scheme would be feasible for the console terminal devices (VT100-LA38) and the RS422 links to the EMM and CI backdoor port in emergency cases. This feature may only be used as an emergency deferred maintenance procedure by field service unless operator can gain physical access (legally) to the cabinet...if so then ensure that the operator procedure is documented in JUPITER operator handbook.

==========

"JUIPTER CPU RAMP"

==========

In this document we will refer to tne KC10 CPU in it's maximum configuration. This configuration contains four functional units: tne IBOX, EBOX, MBOX, and APA. The Memory will be considered in minimum configuration (512K words), and maximum configuration (8M words), utilizing 64k parts. In the event a 256K part becomes available the memory can be expanded to 32M words of Mos memory.

A.3 CPU RELIABILITY IMPLEMENTATION

There has been a large amount of fault tolerant design already implemented in the KC10 CPU and Memory. There is also a considerable amount of fault tolerant design that is planned to be implemented. Additionaly there are some RAMP issues that MEG, because of it's in-depth knowledge of the Service Groups, is proposing to the functional Engineering Groups to be considered for implementation. The following sections outline, in three stages, (IMPLEMENTED, TO BE IMPLEMENTED, and PROPOSED) RAMP requirements that apply to the JUPITER CPU. Where the reason for the RAMP requirement is not intuitively obvious, a supporting argument is included.

***** IMPLEMENTED *****

The following CPU RAMP features dealing with RELIABILITY have been designed into modules that are near ready to be built.

A.3.1 MVE MODULE (EBOX) <IMPLEMENTED>

The following paragraphs will detail the RAMP features on the MVE module. The purpose of the RAMP features on this module is to; detect failures in critical data paths entering and leaving the module, detect failures in the main Arithmetic Unit path, and to detect bus failures to prevent bad data from propagating throughout the System.

Parity generation occurs at the output of the F BUS < ALU result> and at the output of the AND function of the MOVE path. A parity bit is generated for each byte, composed of bits 0:8, 9:17, 18:26, and 27:35, respectively. Parity checking is performed at the output of the SEL OP1 MUX, SEL OP2 MUX, and L MUX. If a parity error is detected at either SEL OP1 or SEL OP2, the following CLK 0 will set the MVL or MVR INPUT ERROR latch which will a) block the next CLK 2 to SEL OP1, SEL OP2, and ALU, and b) will signal to the error control logic on the INX module to stop all gated IBOX, EBOX, and APA clocks starting with CLK 0 20ns later, and sequentially stopping CLK 1, CLK 2, and CLK 3, in that order. If a parity error is detected at the L MUX, the following CLK 3 will set the MVL or MVR OUTPUT ERROR latch which will a) block the next CLK 1 to the L MUX and leading edge control CLK 1 of the MAC BACKUP write-enables, and b) will signal the error control logic on the INX module to stop all gated IBOX, EBOX, and APA clocks.

The carry generates of the two duplicate ALUs are compared to determine whether an ALU error has occurred. If the results of the two ALUs combined produce an odd count, an error trigger is set at the next CLK 1, followed by a MVL or MVR ALU ERR latch at the following CLK 0. The MVL or MVR ALU ERR signal will cause the INX Error Control to stop all IBOX, EBOX, and APA clocks starting with CLK 0.

A.3.2 SH MODULE (EBOX) <IMPLEMENTED>

The purpose of the RAMP features on this modules is to; detect failures on critical storage (accumulators and register files), and to detect failures in the main Shift path of the CPU.

Parity is generated at the output of the SHIFT MATRIX. Four parity bits are carried with all 36 bit wide data paths of the SHL and SHR modules. Parity is modified at the D and X MUXes to

compensate for the 1 and 2 bit shifts. Parity checking occurs at the output of the MAC REG, X REG, and Y REG.

A parity error at X REG of Y REG will set X REG PAR ERR and Y REG PAR ERR respectively. This error condition will a)block the next clock to the X and Y REGs, and b)signal to the error control logic at INX to stop all system clocks at CLK 0.

System reset will reset L1 REG, X REG FILE, Y REG FILE, and Z MUX to zero with good parity. Error reset will reset the error latches.

A.3.3 ESE AND CRA MODULE (EBOX) <IMPLEMENTED>

The Control Storage in the KC10 is parity checked and a method of recovery by Console intervention is employed. This will allow some number of intermittent faults to occur and not cause the System to crash. Additionally the parity check provides a method of dynamic FRU analysis.

Correct parity is stored into the ESE and CRA RAM arrays upon control storage loading from the console. As the microwords are read out, the portion of the word contained on each module is checked for odd parity. If a parity error occurs, the CTL bits of the module are held, along with the NA DLY latch, which represents the address of the failing word (if the failure occurred in the third or subsequent cycle). If a parity error occurs in the first or second cycle, the console may read the instruction code of the failing instruction by stepping back the SEL I in the IBOX. The fast ESE RAMs are parity checked along with zeros (for unused positions) in the first cycle. A parity error will set ESE MICROCODE ERR of CRA MICROCODE ERR latches. These latches will signal the error control logic on INX to stop IBOX, EBOX, and APA system clocks.

System reset will reset all PC Flags and STATE REG bits to zero. Error reset will reset ESE and CRA MICROCODE ERR latches.

***** TO BE IMPLEMENTED *****

The following CPU RAMP features dealing with RELIABILITY are planned to be implemented into the hardware design.

A.3.4 UNCHECKED CONTROL LOGIC <TO BE IMPLEMENTED>

The KC10 will utilize "Microcode" design (where possible) to reduce the amount of unchecked, hardwired, control logic.

A.3.5 IDL AND IDR MODULES (IBOX) <TO BE IMPLEMENTED>

The purpose of the RAMP features on this module is to; detect failures of data entering into the IBOX from the MBOX, detect internal IBOX bus failures (data only), provide an integrity check of the Arithmetic Paths, prevent overwriting of Accumulators with bad data, and provide focus into the module i.e. Passive scanning of the Program Counter.

The OP2 BUS is parity checked as it leaves IDL and IDR. If bad parity is detected, the contents of the OP2 bus are frozen in the OP2 scan reg and one of the two OP2 ERR latches are set (IDL or IDR) at CLK 0.

When bad data is recieved from the MBOX a flag (also from MBOX) is stored in the OP BUFFER signifying a page failure. This flag, when the OP BUFFER is read to be gated onto the OP2 BUS, will cause a page fail trap to occur if the EBOX last cycle bit is on. If the Operand is to be used immediately by the EBOX, then the storing of the flag, in the OP BUFFER, will be too late. What this means is; data on the OP2 BUS may be bad (as a result of the MBOX sending bad data) and the EBOX will trap with an error. During this immediate mode the MBOX needs to send a flag to the FRU logic to be prioritized with the OP2 error seen by the EBOX. This will assure proper FRU isolation.

During long instructions where memory is read in cycles other than the first cycle, i.e. BLT's, the EBOX microcode may branch on a dispatch of a TAG returning whose source code indicates an EBOX operation, and a separate signal from MBOX indicating page fail (or error flag) will be sent back if the operation fails.

Parity is checked at the inputs of the EA ALU. Parity is generated at the output of the EA MUX and PC ALU. One bit parity per 9 data bits is carried for bytes 1, 2, and 3 of the memory address path; one parity bit is carried for bits 6:8 (address byte 0). If an error has been detected at the L BUS on the MVL or MVR modules, the write into the INDEX AC will be blocked. Duplicate copies of the EA ALU are compared against one another

as an integrity check of the EA ALU operation.

A.3.6 APA ISOLATION <TO BE IMPLEMENTED>

The APA will be provided as a customer option to speed up Arithmetic functions. Thus some form of integrity check for operations taking place in the APA should be implemented. The following RAMP feature will detect some number of failures in the Memory Arrays of the APA

Modulo 3 base numbers are generated from both the multiplier and the multiplicand, and a parallel multiply occurs against these numbers. The results are compared to the modulo 3 generation off the full product to check for a multiply array error <called residue check>. Each array card is separately checked for fault isolation.

A.3.7 TAG MODULE (IBOX) <TO BE IMPLEMENTED>

The RAMP features on this module will assure correct and valid data is passed between internal boxes by utilizing a TAG system. The control store for the IBOX is on this module and is checked. The system VAILD INSTRUCTION logic will be implemented on this module.

Parity from MD MUX 9-12 and 32-35 to form a parity bit for the AC ADDRESS BUFFERS; byte 0 parity from the MD MUX is carried with the instruction code. Parity is generated for TAG bits 0-7 as they exit the TAG module for the M BOX. Parity is checked at the instruction code readouts, at the TAG input (returning from MBOX), at ISET readout, and at IPUT control word readout.

The IPUT Microcode word will have odd parity and be checked upon reading any micro location. The address of the failing IPUT word will be held. If an error is detected in the IPUT microcode, the CONSOLE will need to reload that word. Not all of the IPUT data bits will be held.

The EARLY DECODE (ED) Array will be loaded by the Console. ED bits 0-6 will have parity assigned and be checked upon readout.

The VALID INSTRUCTION logic should have a parity check the I VCODE, OP2 VCODE, NXT VCODE, and OP3 VCODE files. Some of the control of the valid instruction logic will not be able to be checked. This is minimal and should not dramatically impact reliability.

A.3.8 INX MODULE (IBOX) <TO BE IMPLEMENTED>

The INX module contains the hart of the Dynamic Fault Analysis hardware of the CPU. It provides a method of stopping the EBOX and IBOX when a hardware detected failure occurs, backing up the PC to a predetermined point, and retrying the operation at fault. This will greatly improve the availability of the system by reducing the number of crashes due to intermittents.

This module also contains the INDEX REGISTERS for the IBOX. Additionally, this module may house some miscellanous logic. See <PROPOSED> section for RAMP features on these areas.

A.3.8.1 PIPELINE STOP MECHANISM <TO BE IMPLEMENTED> -

A two bit shift reg is serial loaded with the and of LAST CYCLE and L EARLY A each cycle. If the ERROR STOP latch is set at any CLK1 time, the contents of the shift reg (RETRY CODE) are frozen. A decode of the retry code determines how many times the PC has stepped since the error occurred:

RETRY CODE	PC
0	N+1
1	N+2
2	N+2
3	N+3

To retry starting with instruction N, a trap is taken by the microcode, which causes it to branch on the retry code to a subroutine issuing a command to the IBOX to step SEL I back 1, 2, or 3 times, and to reload the PC from the PC BUFFER addressed by SEL I.

A.3.8.2 HARDWARE RECOVERY AND RETRY LOGIC <TO BE IMPLEMENTED> -

When an error has been detected, an error latch is set at the module level. The error condition may then block the next clock to the failing register. The error signal is also sent to the INX module where it is funneled into the error stop latch, which will then stop the next clock 0, 1 , 2, and 3 in succession. A 2 bit code is loaded into the FRU REG which identifies to the console the status of the IBOX SEL I tag, a 4 bit value pointing to the next instruction to be executed. The code is used by the console to determine the number of PC instructions to step back in preparation for a retry of the failing instruction stream. The console may also elect to arm the EBOX AUTO RETRY latch, which will force a microcode trap on an error stop condition. A restart routine will branch on the value of the restart code to a microword containing the appropriate command to the IBOX.

The RETRY CODE is defined as follows:

RETRY CODE	RECOVERY
0	restart from IBIX addr by SEL I-0
1	restart from IBIX addr by SEL I-1
2	restart from IBIX addr by SEL I-2
3	restart from IBIX addr by SEL I-3

The EBOX Microcode retry routine is as follows:

1. TRAP to address 7770
2. Disable Traps
3. Clear EBOX data paths and reset error latches
4. Abort APA operation, clear it's data paths and reset it's error latches
5. Load current AC sets from Master AC
6. Dispatch on retry code and step back SELI history counter
7. Retrieve SELI and PC flags of instruction to be retried
8. Skip on Pipe Error (FRU 9)
   a. if Pipe Error, preset IBOX and go to 9
   b. if no Pipe Error, go to 10
9. Read PC of SELI, initialize IBOX to that PC (LD PC command)
10. Enable Traps
11. Issue Last Cycle

Successful retrys appear to be possible for nearly all instructions with only 1 result to be stored. However in instructions with multiword results which overlay previous operands, successful retry is not probable.

A.3.9 MBOX GENERAL <TO BE IMPLEMENTED>

All major buses between the Mbox modules will contain parity check bit(s). Parity will be generated/checked going off each module and is checked again on the receiving module within a few gates. Parity will be checked at the data input to all rams before they are written. The control rams for all of the micro machines will also be parity checked.

A.3.10 CACHE MEMORIES (MBOX) <TO BE IMPLEMENTED>

The Page Table and the Cache Directory have a parity check, while the Data Cache has ECC. When accessing the Cache, a parity error in the Page Table or the Cache Directory generates a page fail condition. A single bit error in the Data Cache is automatically corrected and the syndrome, address, and data are saved for statistical use. It has been agreed that in order to avoid a performance hit, double bit error detection in the Data Cache will not be implemented. However, when multiple single bit errors are occurring, the Console, via command from the Operating

System, will disable the effected Cache quarter. This will greatly reduce the probability of a double bit error occurring.

A.3.11 MOS MEMORY <TO BE IMPLEMENTED>

Each memory array board is organized as 512K words by 22 bits. Each read access to a word of memory produces 44 bits of data from the two array boards, with each 22 bit half word including a parity bit for that half word. Each write to memory also generates 44 bits organized as two sets of 22 bits, with half word parity in each set. On read cycles the 44 bits are sent to the Data Mover, where the half word parities are checked and the ECC is recomputed. Single bit errors are corrected and operation is continued. Double bit errors are detected and a Error flag is sent to the Console. Upon occurrence of a detected error, the address, data, and syndrome of the failing access are latched in a register which is accessible by the Console for error logging. Successive single bit errors destroy earlier data stored in the latch so that only the last error is available in the latch. On occurrence of a double bit error, the error information is stored in the latch, and further single bit errors are corrected but not recorded until the Console logs the pertinent information on the Double Bit failure.

***** PROPOSED *****

The following CPU RAMP features dealing with RELIABILITY are being proposed to be designed into the hardware.

A.3.12 SCA MODULE <PROPOSED>

The RAMP features of this module should be designed consistent with the other CPU modules. Some of the following considerations should be addressed.

A.3.12.1 REDUNDANT ALU'S <PROPOSED> - The Shift control ALU should be redundant, as on the MVE module, and the carries of the ALU's compared for failures.

A.3.12.2 TIMERS AND METERS <PROPOSED> -

There should be a method to assure reliability of the internal timers and meters. Where possible parity should be generated on initial load, updated after count duration, and checked prior to reading the meter into the CPU data paths. If parity cannot be assigned and checked, (due to timing and the amount of logic required to implement this) then at initial load (preset) an optional "count then scan" test should be implemented at micro level. This could be enabled on installations where accounting is very critical for billing, or by Field Service when a problem is suspected in this area.

A.3.13 INX MODULE <PROPOSED>

Additional to the FRU logic that is to be implemented on the INX logic, there are several other areas of concern that should be noted.

The INX module will contain the index registers for the IBOX. The following IR's will be on this module and should have parity assigned and checked upon readout; the OP2 INDEX, OP3 INDEX, AC INDEX, and the XR. This module may also house some miscellanous logic that may overflow from other modules. If this is the case, the overflow logic should be checked in a consistent manner with the rest of the CPU.

A.3.14 APA DISABLE <PROPOSED>

Since the APA is a logically extension of the EBOX, and is a Customer option, there should be a method of defeating it's use. This would allow degraded system operation when a fault occurs. If possible the disabling should be accomplished dynamically without reloading the Operating System.

It appears that this disabling is possible, and can be accomplished by; the Console reloading some microcode in the IBOX/EBOX and continuing the System. We will need to determine if the Console will be able to tell what locations in the IBOX/EBOX it needs to reload and what to reload them with. We will also have to determine what the Monitor will need to do (if anything) to allow this to happen.

A.3.15 HUNG CONDITIONS <PROPOSED>

There should be a method to detect hung conditions, (incomplete cycles) or tight micro-loops in the CPU. This could be implemented for areas where critical timing is involved. The effected logic could be reset and retried. Obviously not all hung conditions will be able to be detected, and there may also may be times when tight micro loops are normal. However, there are certain cases where operations are rather synchronous and are limited in the amount of time taken until the operation completes e.g. the address of a memory request is placed on the bus, and after some maximum amount of time, an ACKN should come back, if not a time-out occurs and the result is a NXM. With a microcoded machine other (not so discrete) critical events could be timed for maximun allowable duration. This may be especially desirable in the MBOX where the events are more synchronous. Having this focus could eliminate some of the infamous "KEEP ALIVE CEASE" problems, or at least give some insight on what caused it when one does occur.

A.3.16 SYSTEM CLOCK CONTTROL <PROPOSED>

Since the clock system of the KC10 is the hart of it's operation failures in this area would be very catastrophic, especially failures that may go undetected. With this in mind we need to assure high reliability in the clock subsystem. The following should be considered as RAMP features relating to reliability:

A.3.16.1 PHASE CHECK - The CPU requires four phases to complete one machine cycle. A malfunction in any of these phases will probably be catastrophic to system operation. For this reason we need a RAMP feature that assures one and only one phase is active at any one time. Additionally, if implementable, there should be a method to detect missing phases and stop the clocks at the next

active phase.

A.3.16.2 BACKPLANE DISTRIBUTION - In the past many of the apparent clock problems were due to faulty distribution coax. With the increased speed of the KC10, the method of clock distribution becomes even more critical. For this reason we need to have a highly reliable distribution network (preferably etch). The reliability of this network is difficult to quantify. However, the backplane etch or coax utilized for clocks should be as reliable as all other backplane interconnects.

A.3.17 MBOX ERROR RECOVERY <PROPOSED>

Where possible, error recovery should be preformed at microcode level. Pertinent error information should be saved and made available to the Console and Operating System for logging into SPEAR.

A.3.18 MOS MEMORY ADDRESS PARITY <PROPOSED>

On the detection of an Address Parity error on a main memory write cycle the controller will disable the write enable signal to the arrays, thus forcing the MOS rams to do a read operation rather than a write. This prevents overwriting the wrong location. The normal reaction is to automatically retry the write operation a 2nd time in the event the 1st error was intermittent. This retry should be accomplished at Micro level. If the retry is not successful the MBOX should save the failing address and data and set an address error flag. What the Micro engines, and the Console will do, if the error is solid, needs to be determined. How the Monitor will handle a solid error, assuming the Monitor stays alive, needs to be determined.

A.3.19 MBOX INTERNAL ADDRESS PARITY <PROPOSED>

Parity is carried from the IBOX on VMA data. Address information throughout the MBOX should have parity assigned. The parity should then be checked at the input to the Translation Buffers, Cache Directory, Cache Address lines from the MAP to the MDP, etc.

A.3.20 CACHE USE BIT TABLE AND REFILL RAM <PROPOSED>

If the Data Cache is to utilize a Refill Ram and Use Bit Table, the parity should be assigned and checked. Failures in this area should not be catastrophic to System operation. However, failures should be logged to allow repair as the System will be running degraded.

A.3.21 DATA CACHE WRITTEN AND VALID BIT FAILURES <PROPOSED>

The Written and Valid Bits in Cache Memories should have parity assigned and be checked. There are some concerns on how to handle failures in this area both at Hardware/Firmware and Operating System level. However, the Hardware/Firmware should be implemented to detect failures.

A.3.22 PAGER CACHE VALID AND KEEP BIT FAILURES <PROPOSED>

The Pager Cache should have parity assigned to the Valid and Keep bits. This parity should be seperate from the Translation Data. This will assure the continuation of the proper parity when the Valid or Keep Bits are toggled.

A.3.23 LOSS OF MBOX OR IO CLOCKS <PROPOSED>

The MBOX and IO clocks are to be free-running. If a failure occurs in the Clock logic, or the clocks are stopped due to a fatal error i.e. MBOX Microcode parity error, there is a good chance IO overruns will occur. If this happens on IO reads the result can be File Corruption. If the failure is intermittant, and the System does not crash or is successfully restarted, the IO request should be retried in order to correct the data on the disk. The retry should be done at Micro level if possible. It is more probable that the IO will need to be retired by the Operating System. If this is the case, then the hardware (MBOX, PORT, IOBOX, CONSOLE, ???) must inform the Operating System of the overrun so it knows it needs to retry the operation.

A.3.24 BAD IO DATA <PROPOSED>

Bad data with bad parity entering the MBOX that is a result of IO errors should not be stored in Memory. If this cannot be accomplished then the retry algorithm must assure the bad data does not remain in memory to be retrieved later. Bottom line, IO Errors should not lead one to believe there is Memory or MBOX subsystem problems.

A.4 CPU MAINTAINABILITY IMPLEMENTATION

The following sections provide RAMP requirements that address Maintainability. Special emphasis is put on, the realization that intermittents do occur, how to recover from them, and how to troublesnoot them when they cause failures. The requirements are categorized in three stages: IMPLEMENTED, TO BE IMPLEMENTED, and PROPOSED. Where necessary supporting reasons are given.

***** IMPLEMENTED *****

The following CPU RAMP features dealing with MAINTAINABILITY have been designed into modules that are near ready to be built.

A.4.1 OVERALL ERROR CONTROL AND DETECTION LOGIC <IMPLEMENTED>

Error control logic for KC10 is designed in such a way as to support 3 major RAMP goals; A) detection and recording of intermittant as well as solid failures, B) hardware recovery from a high percentage of CPU errors, C) fault isolation <to a module> of a high percentage of errors prior to running of diagnostics and, in the case of intermittants, without taking the system down. A high emphasis is placed on: "catching" intermittant errors as close as possible to the point of failure. 20 to 25 % of JUPITER logic is devoted to the implementation of these features.

Strategic registers are frozen if an error is detected at their output. This approach helps in locating the source of an error likely to be propagated through the system or lost, by the time the IBOX, EBOX, and APA module clocks can be stopped.

A.4.2 DIAGNOSTIC INTERFACE <IMPLEMENTED>

The diagnostic interface is the vehicle through which the console A) presets execution unit registers to desired states, B) forces control decodes to manipulate the data paths, C) gains visibility into the data and control paths, and D) loads control storage areas. The 100141 shift register is used to achieve all of these. This provides the ability to use the scan in/scan out mechanism as a functional part of the data path under normal system operation and as a diagnostic path under Console control. All shift registers used in this manner and strung together (one's shift output connected to the other's shift input) are considered to be part of the active scan path. Other shift registers, not part of functional logic, are used for presetting signals to desired states, and scanning data path and control states independently of E unit operation. These are linked together as the passive data path.

A.4.2.1 ACTIVE SCAN PATHS <IMPLEMENTED> -

The KC10 CPU contains five active scan paths. Each is sourced at the IO BOX and consists of shift registers which, under normal operation (not clocks stopped due to error or console control) perform a parallel load when their corresponding system clocks occur. When the system clocks have been blocked by

error freeze or a clock control stop, the console, via the IO BOX may perform one of two operations; A) a parallel load via its own clock, B) a serial shift (normally hi-order to lo-order).

A.4.2.2 PASSIVE SCAN PATH <IMPLEMENTED> -

The KC10 CPU contains two passive scan loops. One passive scan loop is dedicated to the CLK CONTROL module, while the other encompasses all other KC10 CPU passive controls and readouts. A command ENABLE from the console will be used to gate those bits which are used as presets or RAM write-enables once the desired section of a passive loop has been loaded.

A.4.2.3 SCAN READ AND LOAD <IMPLEMENTED> -

A 16 bit shift register in the console completes the scan path loop. The register is loaded at the console through a 16 bit mask, thus permitting individual bits to be loaded without altering the masked bits.

A.4.2.4 DIAGNOSTIC SCAN PATH VISIBILITY <IMPLEMENTED> -

The scan-in, scan-out mechanism described above is the method by which console may assert control or data path signals, and read out a high percentage of the data path. The 100141 shift registers constitute approximately 12% of JUPITER execution unit logic. Diagnostic control has reserved to it a section of the microcode control storage, thus allowing it to route, via set up of next address to read a microword with desired control decodes, virtually all registers (if not already a 141) through mux paths and into 141s for scanning. This includes access to 16 word register files and RAM arrays.

***** TO BE IMPLEMENTED *****

The following CPU RAMP features dealing with MAINTAINABILITY are planned to be designed into the hardware.

A.4.3 FAULT ISOLATION BY INTERNAL HARDWARE <TO BE IMPLEMENTED>

The following is a planned implenentation of hardware FRU isolation logic in the EBOX, IBOX, and APA.

The FRU logic will be used to accomplish dynamic fault isolation, and will prioritize multiple errors. This will provide a rapid MTTR for a large portion of CPU errors. Additionally it will provide a method to assure logging of intermittent errors that otherwise may have gone unnoticed due to the hardware retry logic.

THE FRU REG

UCODE ERR	FRU1	FRU2= FPI	RETRY CODE
0:8	9:14	15	16:17

For each error stop, the FRU REG is loaded at the INX module, provided that the FRU STORED F/F is off. The FRU REG is 18 bits in length and identifies to the console the 1st and 2nd most probable failing modules, and whether a failing module containing reloadable storage requires a transfer from backup storage. The errors are prioritized so that the source of the error will be identified, rather than the path which merely propagated or received erroneous data. Since the errors are frozen dynamically, and fed through a priority encode to generate the FRU CODE, an accurate method of trouble-shooting one error at a time is achieved. A 6 bit code, labeled FRU1, identifies the most probable failing module. FRU2 identifies the 2nd most probable failing module to be the FPI module, if an MVL or MVR output error has occured, and RF and L MUX controls have selected the A BUS to the output of the L MUX. Bit 9 of the FRU REG identifies a possible pipeline failure. During the microcode recovery routine, if this bit is on, the EBOX will initiate a flush of the IBOX pipeline. Bits 0 through 8 of the FRU REG indicate that reloadable storage has possibly failed in one of the following areas:

FRU REG 0	IPUT microcode	TAG module
FRU REG 1	ED table lookup	TAG module
FRU REG 2	ISET microcode	IDL module
FRU REG 3	ESE microcode	ESE module
FRU REG 4	CRA microcode	CRA module
FRU REG 5	SHL MAC	SHL module
FRU REG 6	SHR MAC	SHR module
FRU REG 7	FPI microcode	FPI module
FRU REG 8	Divide Lookup	FPI module

A failure at reloadable storage will block Auto Retry, even if console has enabled the Auto Retry mechanism. Console, after reloading the failing area, must force the EBOX microcode to enter its retry routine via the CRA load address reg.

Each of the above areas has a backup. In the case of a microcode or divide lookup error, the diagnostic load address reg for that storage contains the failing address; therefore console may reload that word with data from it's own microstorage backup. In the case of a MAC error, the hi order 4 bits of the write address are held at the MAC backup on MVL and MVR, and will be used by the console to reload the failing 16 word set from MAC backup.

Below is a list of errors versus FRU codes in order of priority.

ERROR	FRU 9:14
IPUT MICROCODE ERROR	40 TAG
ED TABLE LOOKUP ERROR	41 TAG
TAG IN ERROR	42 TAG
LEFT E1E2 ERROR	43 IDL
RIGHT E1E2 ERROR	44 IDR
LEFT EA ALU ERROR	45 IDL
RIGHT EA ALU ERROR	46 IDR
SPARE	47
ISET ERROR	50 IDL
ICAC ERROR	51 IDL
SELI ERROR	52 TAG
LEFT OP2 ERROR	53 IDL
RIGHT OP2 ERROR	54 IDR
SPARE	55
SPARE	56
SPARE	57
LEFT REG OR AC ERROR	10 SHL
RIGHT REG OR AC ERROR	11 SHR
FPE AC ERROR	12 FPE
FPD AC ERROR	13 FPD
FPC AC ERROR	14 FPC
FPB AC ERROR	15 FPB
FPI OP ERROR	16 FPI
SPARE	17
MVL OUTPUT ERROR	20 MVL
MVR OUTPUT ERROR	21 MVR
MVL ALU ERROR	22 MVL
MVR ALU ERROR	23 MVR
SCA OUTPUT ERROR	24 SCA
RETURN ERROR	25 CRA
MVL INPUT ERROR	26 MVL
MVR INPUT ERROR	27 MVR
SCA INPUT ERROR	30 SCA
SPARE	31
SPARE	32
FPE RESIDUE ERROR	33
FPD RESIDUE ERROR	34
FPC RESIDUE ERROR	35
FPB RESIDUE ERROR	36
FPA RESIDUE ERROR	37

NOTE

A LOW PRIORITY INPUT FOR A MARK BIT WILL BE ADDED TO THE FRU LOGIC. AN INPUT FOR MBOX ERROR WILL BE ADDED...SEE IDL AND IDR SECTION PARAGRAPH THREE FOR DETAILS

A.4.4 MEMORY RECONFIGURATION <TO BE IMPLEMENTED>

Memory reconfiguration to the smallest operational part must be able to be accomplished with minimal System down time. This should be accomplished easily, via the Console, by use of a Humanely Engineered command.

A.4.5 CONSOLE LOGGING OF MBOX ERRORS <TO BE IMPLEMENTED>

For a majority of classes of errors, in the MBOX, the Console will have prime responsibility for capturing initial error status. This will allow the Mbox to continue operation and still have the error information logged. The Console will pass this error infromation to the Operating System for logging into the Error File, assuming the Operating System remains alive. If it doesn't, the Console will hold the Error information (in non-volatile storage, or on media) for retrieval, while the system is down, and also to allow passing of the information to the Operating System when it is reloaded.

A.4.6 MOS MEMORY MODULE ISOLATION <TO BE IMPLEMENTED>

Failures in MOS Memory should be isolated to a single module 100% of the time. This is necessary do to the high cost and the density of the module. In the past we caused the customer to suffer through multiple System crashes while we isolated the failing stack in KL10 Memory. There must be a method, by use of Hardware/Fireware, or at least via the Console to isolate to the half word. This must be possible under the Operating System without taking the system Standalone or by the use of Exec Mode Diags.

A.4.7 SYSTEM CLOCK CONTROL <PROPOSED>

Within any machine/logic family, there is a class of failures are timing sensitive. They usually surface on an intermittent basis. Running diagnostics, or even the Monitor at the normal system clock speed does not always yeild a failure rate that will allow isolating these failures. For this reason, we need to design a Clock Control mechanism that gives us flexability and control of all clock parameters. This control will be utilized by diagnostics to isolate failures. Field Engineers will use this feature to force higher failure rates and aid in troubleshooting of intermittents. The following are some RAMP requirements that we will need to assist in maintainability of the KC10: (Also see RELIABILITY section for more information on the CLOCK SYSTEM).

A.4.8 CONTROL BY CONSOLE

The CONSOLE needs the ability to gain total control over the CLOCK SYSTEM. This will be utilized by the CONSOLE to START/STOP the Clock System at system load time, during diagnostics, to passively read registers, and handle error recovery and reporting. Some of the features of this Control Mechanism are as follows:

A.4.8.1 FREQUENCY CONTROL -

To facilitate the troubleshooting of the above mentioned intermittents, we need to be able to margin the normal clock speed, (master oscillator frequency) by some percentage upward and downward from the optimal setting. This percentage should be defined by the tollerance of the logic being margined. If it takes a 10% margin to cause CKECL to fail at a higher than normal rate, then we should be able to margin the clocks to this point. Additionally a slow down of the clock speed may be needed to facilitate use of extender boards.

A.4.8.2 ENABLE DISABLE FEATURE -

The CONSOLE must have the ability to enable and disable various system clocks. This will allow intervention on CPU errors plus give concise control during diagsostics. The enable disable control should be segmented so that one portion of the machine can be stopped without effecting another i.e. EBOX, IBOX, FPA clocks should be stoppable without effecting the MBOX and IO clocks. When the disable command is initiated the clocks should be stopped at the next Phase, or as soon feasibly possible. This will prevent unwanted data and control signal propagation.

A.4.8.3 TIC AND PULSE MODES -

To provide diagnostics with the ability to single instruct (micro and macro level) the machine there should be a method to TIC the CPU clocks. The TIC interval should be at least as small as one machine cycle (four phases). It may be benificial to have a sub-machine level TIC that would step by phases.

There should be a method to step the machine at pulse intervals. What is desired here is the ability to load a counter and run the clocks for some number of machine cycles. Preferably the number of counts loaded by the CONSOLE. The Pulse interval (loadable counts) should be at least as large and the single most complex machine operation. This would certainly not cover all cases of long instructions i.e. BLT's, but would assure total pulse control over most macro instructions.

.P
The following CPU RAMP features dealing with MAINTAINABILITY are being proposed to be designed into the hardware.

A.4.9 ERROR PROPAGATION AND MULTIPLE ERRORS <PROPOSED>

In the past one of the most difficult failures to isolate was the multiple error condition. This was caused many times by allowing bad data to propagate through the machine. In order to keep the list of possible FRU's to a minimal when an error occurs we are requesting that errors be captured as close to the source as possible and not allowed to propagate. Also when local retry is utilized all bad data must be logged then cleared from registers prior to restarting the machine. This will prevent erroneous and redundant errors.

A.4.10 ERROR INFO RETRIEVAL ON A "DYING/DEAD" MACHINE <PROPOSED>

There will be a failure mode where the retry algorithms are unsuccessful or the retry count is exceeded and the System crashes. Yet, when the isolation diagnostics are run no failure will occur. This failure mode is by far the most difficult to troubleshoot. Our past experience tells us we need a good hardware and software SNAPSHOT of the machine when the failure occurs. The software snapshot (dump) will be taken by the Monitor, as traditional. However, there is still times when one cannot get the Monitor up long enough to analyze the crash. For this class of errors we are requesting a Console SNAPSHOT of all the Error signals and pertinent Control logic. This Snapshot should happen as close to the time the error occurs (assuming it is not recoverable). System and Error Reset signals must not clear registers and error information before this Snapshot occurs. The Snapshot should be saved in nonvolatile storage or on the Console Disk. There should also be a method to retrieve this information and have it formated in a logical and easily readable manner.

A.4.11 PIPELINE OVERRIDE <PROPOSED>

With the circuitry complexity that will be involved with the introduction of the PIPELINE, there should be a method to defeat it's use. This would enable Field Service to isolate intermittant failures that were caused by interaction due to Pipelining.

A.4.12 MBOX HARDWARE FRU ISOLATION <PROPOSED>

If possible the HARDWARE FRU logic should have an input for MBOX errors.

Since the MBOX does not utilize the same clock stop mechanism as the E/I BOXES, the retry procedure whould not be valid. However, there should still be a method to "HARDWARE" priortize, and log MBOX errors. This may have to be a seperate circuit from the E/I BOX FRU logic. If so, a retry algorithm by MBOX Microcode could be developed for a large portion of errors. This would reduce the amount of recovery work the Operating System would have to do. Since the retries would be done at Micro Level, intermittents may go virtually unnoticed by TOPS. Additionally since the retry is at Micro Level, not Macro Level, (as TOPS would be) the probability of it being successful may be higher. This may not apply to areas with ECC.

A.4.13 MBOX ERROR LOGGING <PROPOSED>

For most classes of errors in the MBOX, the Console should be the prime path for error logging. However, the Console should collect the data and pass it on to the Operating System for logging and analysis by SPEAR. Where this method is not possible, traditional logging procedures should be applied. This will probably be in the areas of IO and the Translation Buffer (Page Faults). Any additional information that may be useful to the Operating System for recovery, or to Field Service for maintenance, should be captured via the various Micro-Machines, and saved in scratch pad for retrieval.

A.4.14 PC TRACE MODE <PROPOSED>

There should be a PC trace mode enable function. This function would allow for the tracing of the last few "previous PC's" where a previous PC is the PC as a result of the last skip or jump instruction (including JSYSs, UUOs, PIs, and TRAPs). Preferably at least 64 previous PCs should be available. It would then be possible for Field Service or Software Service to enable this mode in order to track down an obsure (but repeatable) hardware/software porblem.

A.4.15 ADVANCED ADDRESS BREAK FACILITY <PROPOSED>

There should be a more advanced address break facility than the current single breakpoint location. This facility should provide for a range of addresses to be supplied and some kind of masking, both on data and addresses. The addresses should include section numbers as well as the standard 18 bit address.

"IO RAMP"

The I/O scheme requirements are in general, to provide utmost data integrity; upon failure, to contain the error to the failing module, to automatically recover with minimal operating system intervention and impact; and finally, provide personnel (both hardware and software) with all information necessary to perform optimum service.

Each area of the I/O section is broken into the specific hardware section for ease of reading and direction to interested parties.

A.5 IOBOX RELIABILITY IMPLEMENTATION

A.5.1 INTERNAL PARITY <TO BE IMPLEMENTED>

A parity scheme must be implemented on all data and control signals entering and leaving each module. The network should be partitioned such that isolation to the single failing module is achieved.

A.5.2 IOBUS ARBITRATION <TO BE IMPLEMENTED>

IOBUS arbitration errors must be detected and stored; also, a mechanism must be implemented such that unique PORT data integrity is maintained. (ie. PORT # 2 does not process data destined for PORTS # 0,1,3 etc.) This also implies some responsibility in the PORT.

A.5.3 INCOMPLETE CYCLES <TO BE IMPLEMENTED>

An incomplete cycle or a hung condition must not cause a global system failure. An individually controlled logic reset, available to the PORT for the memory<----->I/O path, and also to the console for the KC10 support path, will enable retries to be effected. A report should be available for SPEAR, describing the error and number of retries necessary.

A.5.4 MTBF <TO BE IMPLEMENTED>

The IOBOX has the potential to be a "single point of failure"; therefore it must be highly reliable, ie. "MTBF greater than 10,000 hrs".

A.6 MAINTAINABILITY IMPLEMENTATION

A.6.1 ERROR RECORDING <TO BE IMPLEMENTED>

The IOBOX must be able to detect, store via a bus recorder, and report on demand(from the MAINTENANCE CONSOLE or PORT), all errors in a consistant and efficient manner.The Error Data must be saved for analysis by SPEAR.

A.6.2 LOOPBACK <TO BE IMPLEMENTED>

There must be internal and external loopback paths for all interfaces.

A.7 PORTS

Although the PORT must interface to four or more unique ADAPTORS/LINKS, there are certain generic requirements that apply to all schemes.

A.8 RELIABILITY IMPLEMENTATION

A.8.1 INTERNAL PARITY <TO BE IMPLEMENTED>

A parity scheme must be implemented on all data and control signals entering and leaving the PORT.

A.8.2 MICRO-CODE ERRORS <TO BE IMPLEMENTED>

Microcode parity errors must be reported and recovered. It is envisioned that the PORT will flag the MAINTENANCE CONSOLE (via the scan path) for help.

A.8.3 EXTERNAL ERRORS <TO BE IMPLEMENTED>

The PORT must be able to handle and report TTL-BUS and IOBOX errors.

A.8.4 INCOMPLETE CYCLES <TO BE IMPLEMENTED>

Incomplete/incorrect interface cycles must be be detected, retried, and reported.

A.8.5 DATA INTEGRITY <TO BE IMPLEMENTED>

The PORT must periodically, or on demand from the operating system, verify data mover integrity for all data packing modes. A parity network is not sufficient to guarantee that "data in = data out" because parity is not computed until after the data manipulation is complete. Thus a malfunctioning data mover would not be detected until a diagnostic program were run which will be long after the point of failure. By this time the system is a disaster (many files destroyed) and a system file restore is necessary. The worst case scenario is of course, this type of failure while the system is being backed up. Two things could be done to prevent this from happening:

1. A periodic data mover check (using a known data pattern) could be initiated by the microcode.

2. The operating system could initiate the ucode test at the appropriate time. (eg. BACKUP/RESTORE, CHECKD, etc.)

A.8.6 CLOCK MARGINING <TO BE IMPLEMENTED>

If clock margining is anticipated, any valid margin selected must not trash data on the device. Since we don't have a spec for the clock scheme yet, it is difficult to ascertain whether we wish to pursue this issue further.

A.8.7 MAINTAINABILITY IMPLEMENTATION

A.8.8 DISABLE FEATURE <TO BE IMPLEMENTED>

The capability to disable any PORT must be provided so that no PORT can cause a global system failure. This can be accomplished by directing the MAINTENANCE CONSOLE to put the PORT in a (yet to be defined) halt loop (if possible), or to disable the PORT clocks.

A.8.9 LOOPBACK <TO BE IMPLEMENTED>

Diagnostic loopback paths are required for all PORT interfaces.

A.8.10 POWER-UP STATUS CHECK <TO BE IMPLEMENTED>

At power-up/start-up time, the PORT should provide an internal control check and an external interface check. It must also be capable of reporting it's configuration and status. A proposed method for handling this feature is to have each type of adaptor generate a unique 3-bit code. The MAINTENANCE CONSOLE, through it's PORT shift register interface, will then be able to determine the hardware configuration and load the appropiate ucode.

A.8.11 ERROR RECORDING <TO BE IMPLEMENTED>

Data in error must be saved and made available for analysis by SPEAR. This will help to reduce the time required for module repair (ie. For intermittants, much time is wasted in attempting to establish the mode of failure) by having real-time error data accompany the module to the repair center. We will specify that a procedure to support this requirement be included in the maintenance strategy.

A.9 ADAPTORS

Only generic requirements for all adaptors will be listed here, specific features which are applicable to a unique scheme will be found in that section (ie. MBA,UBA etc.).

A.10 RELIABILITY IMPLEMENTATION

A.10.1 INTERNAL PARITY <TO BE IMPLEMENTED>

Implementation of a parity scheme to detect, report, and contain erroneous data. In the past, these functions were not always uniform in their implementation, which caused many headaches for the field, both in diagnosing the failure as well as being an unecessary burden on logistics (eg. 3 modules are ordered "P1" to resolve a problem when 1 module would suffice, given proper diagnostic module isolation).

A.10.2 INCOMPLETE CYCLES <TO BE IMPLEMENTED>

Incomplete/hung cycles must be detected, retried, and reported. It would be possible for SPEAR to collect and analyse this class of failures; and then provide guidance for the service group to schedule deferred maintenance.

A.11 MAINTAINABILITY IMPLEMENTATION

A.11.1 LOOPBACK <TO BE IMPLEMENTED>

Loopback paths for internal checks and external bus testing to be invoked for diagnostics and also upon command from the operating system/SPEAR. This will be useful for the case where anaylsis by SPEAR cannot conclusively isolate a failing function. Say for example, that more failure data is needed, but media for testing is not available or required. SPEAR could then invoke this type of internal test to produce the failures required to complete the anaylsis.

A.11.2 DYNAMIC DISABLE <TO BE IMPLEMENTED>

The adaptor must have the capability to be disabled if a solid fault condition exists. This should be able to be done dynamically and not cause the system to crash if the Adapter is not required for system operation. The PORT disable feature would seem appropriate here.

A.11.3 RAMP PARTITIONING <TO BE IMPLEMENTED>

Parity networks should be positioned to allow optimum board isolation. They should take into consideration both physical positioning(as close to the module edge as possible) and logical position with respect to real time detection and reporting(i.e. by the operating system/SPEAR).

A.11.4 ERROR RECORDING <TO BE IMPLEMENTED>

Data in error must be saved and made available for SPEAR anaylsis. This will greatly reduce the time required for module repair (ie. for intermittants, much time is spent to establish the failure mode with diagnostics) by having real-time error data accompany the module to the repair center. We will specify that a procedure to support this requirement be included in the maintenance strategy.

A.11.5 ON-LINE MAINTENANCE <PROPOSED>

The technique to perform corrective maintenance without impacting or stopping the system (i.e. replacement and verification of a known defective module) is believed to be easily attained. However, our perception is that LSG customers will opt to buy redundant hardware to gain increased availability rather than chance overall system failure due to human error if this technique were employed. Therefore we will withdraw this request.

A.12 MASSBUS ADAPTOR

A.13 RELIABILITY IMPLEMENTATION

A.13.1 INTERNAL PARITY <IMPLEMENTED>

A parity network is provided for all data and control signals entering and leaving the MBA.

A.13.2 MASSBUS INTEGRITY <IMPLEMENTED>

Failures in MASSBUS dialogue must be detected, retried and reported. An example would be "register access error" or "data bus time-out" dialogues. Given the poor track record of the MASSBUS, indicators such as these will provide advance warning that the bus is degrading. PORT microcode should pass these errors to the operating system for subsequent processing by SPEAR.

A.13.3 DATA INTEGRITY <IMPLEMENTED>

In the event of an I/O path failure (excluding MBOX clock stop) during a disk write command, the MBA in concert with the PORT, must continue to transfer all available data until the end of the current sector (EBL), and then terminate and report error status to the operating system. It is then the responsibility of the operating system to ensure that the record in error is re-written correctly.

A.14 MAINTAINABILITY IMPLEMENTATION

A.14.1 LOOPBACK <IMPLEMENTED>

Diagnostic loopback paths are required for all interfaces.

A.14.1.1 DYNAMIC DISABLE -

The capability to disable the MBA is required so that no MBA can cause a global system failure. This should happen dynamically and not cause the system to crash, assuming the effected MBA is not required for system operation. The PORT disable feature is acceptable.

A.14.2 INCOMING PARITY CHECK <IMPLEMENTED>

This network has been moved as requested and it will provide a check on the output ram to enhance failure isolation by SPEAR (using live data). A detailed justification for this is not possible at this time because the reliability calculations have not been completed for the circuits in question. (74LS298-4x2 mux,AM29705-dual port ram)

A.14.3 ON-LINE MAINTENANCE <PROPOSED>

The technique to perform corrective maintenance without impacting or stopping the system (i.e. replacement and verification of a known defective module) is believed to be easily attained. However, our perception is that LSG customers will opt to buy redundant hardware to gain increased availability rather than chance overall system failure due to human error if this technique were employed. Therefore we will withdraw this request.

A.15 PACKET BUFFER

A.16 RELIABILITY IMPLEMENTATION

A.16.1 INTERNAL PARITY <TO BE IMPLEMENTED>

A parity network for both the transmit and receive data paths is required to provide stated module isolation goals. They should also conform to the standard of detecting but not propagating erroneous data.

A.17 MAINTAINABILITY IMPLEMENTATION

A.17.1 LOOPBACK <IMPLEMENTED>

An internal loopback path is provided for diagnostic use. This allows data path problem resolution to be no more than 4 chips.

A.17.2 DIAGNOSTIC FOCUS <IMPLEMENTED>

A diagnostic mixer, connected to the "PORT-LINK-INTERFACE", can be enabled by a maintenance mode command from the port. In this mode, 32 internal signals can be sampled for further definition or isolation of a problem.

A.18 UNIBUS ADAPTOR (TBD)

A.19 BLOCK MULTIPLEXER (TBD)

APPENDIX B

JUPITER MAINTENANCE SOFTWARE RAMP REQUIREMENTS

The diagnostic ramp requirements pertaining to JUPITER address two areas , reliability and maintainability. The requirements related to reliability address the ability of the system to meet its aggressive availability goals thru fault tolerant hardware and software design. Diagnostic Engineering's prime responsibility in the reliability area is the console software product which supports those goals . The focus in the reliability section is the transient or intermittent fault from which recovery is possible. The Diagnostic maintenance software ramp requirements are addressed in the maintainability section. The maintainability section focuses primarily on the solid fault, defined as a reproducible error event that the system is unable to recover from or circumvent in order to continue running the customer's application. The diagnostic ramp requirements in that section focus therefore on returning the system to normal operation from a remedial maintenance situation.

B.1 DIAGNOSTIC/CONSOLE RAMP- RELIABILITY IMPLEMENTATION

The console software contributes to the availability goals of the JUPITER by supporting the fault tolerant design of the system.

The primary components of the console software product are:

KCROM-basic hardcore tests always resident
KCCON-loadable console code
KCRSP-run-time support package

***** IMPLEMENTED *****

B.1.1 CONSOLE RELIABILITY <IMPLEMENTED>

***** TO BE IMPLEMENTED *****

B.1.2 DUAL CONSOLE LOAD DEVICES <TO BE IMPLEMENTED>

Dual copies of all microcodes, pre-boots ,recovery algorithms for which the console is responsible and error threshold parameters, one copy for normal use, the other copy on the other console media as backup.This reduces the system single point failures to the common controller (RLV11). I.E. If the first RL02 is not operative at load time OR becomes inoperative during system operation, then the second RL02 will be auto-selected as the prime console resource.

A scheme to safeguard as reasonable as possible the critcal KC10 fault isolation diagnostics resident on the first RL02 . (See proposed reliability section under RL02 for more detail.)

B.1.3 VT100 <TO BE IMPLEMENTED>

As a trade off for the removal of the alpha-numeric display system, a split screen system has been committed to by engineering. The top section will be used for normal operator/system communication. The bottom section will be used by the RUNTIME SUPPORT PACKAGE. Reliability wise, the advantage will be to provide early warning to the operator or Field Service person of problems requiring attention such as: environmental parameters exceeded, severe system degredation occurring due to error handling loops and other. The messages have not yet been enumerated out as a general guideline, they shall be concise, clear and in English. Critical messages shall blink to ensure attention. If the operator is required to act, the options might want to be spelled out via a help library.

B.1.4 RL02 PROTECTION SCHEME <TO BE IMPLEMENTED>

To insure a one console disk runtime environment should one RL02 become inoperable, and still provide a measure of file security for the critical diagnostic disk a dual home block/dual directory scheme has been suggested. This is a modified DOS file format coupled with an unconditional read after write to minimize file damage and allow for file writes during the single drive scenario (minimum reduction of ramp features).

It is a stated non-goal to support usage of the console load device as a system media for file interchange. i.e. The RL02's are not system devices.

B.1.5 JUPITER DIAGNOSTIC PACKAGE <TO BE IMPLEMENTED>

While all of the final code pieces have not been mapped out, the following list represents the current plan to meet the reliability requirements mentioned:

RL02 #1

KC10 MICROCODE(S)
PORT MICROCODE(S)
PREBOOT
-MONITOR
-DIAGNOSTIC
NV RAM BACKUP
EMM LOAD
CURRENT CONFIF FILE(S)
KCMON RLPIP
KCCON KCFORM
KCTAPE KCTRACK
KCCNV
KCRSP
KCPIP
KCDSP
KC10 FAULT DIAGS
PORT AND ADAPTER DIAGS
6 RL02 DIAGS

***

RL02 #2

KC10 MICROCODE(S)
PORT MICROCODE(S)
PREBOOT
-MONITOR
-DIAGNOSTIC
NV RAM BACKUP
EMM LOAD
CURRENT CONFIG FILE(S)
RLPIP
KCCON
KCAPT

KCRSP
KC10 FUNCTIONAL DIAGS
EXERCISOR DIAGS
KCDNLD AND 11/34 DIAGS

6 RL02 DIAGS AND 11/23 DIAGS
DEVICE DIAGS
-RP06
-RP20
-TU78
-TU72
-DX20
-BLOCK MUX
TEMP FILE SPACE
-ERROR FILE OVERFLOW
-SPEAR ANALIZER BUFFER
-TEMP RSP DUMP SPACE
-FUTURE APPLICATIONS

NOTE

*** TEMP FILE SPACE ON RL02 1
DEPENDANT UPON WRITE PROTECT
DECISION AND SIZE OF KC10 FAULT
ISOLATORS

B.1.6 INTEGRITY CHECK <TO BE IMPLEMENTED>

The console must check the following systems for basic functionality at system startup:

Console
CPU
Memory subsystem
Disk Channel
TOPS-10/TOPS-20 paging

The console will perform an integrity check of the system device boot path as close as possible to the actual media prior to initialization of system boot procedures. In order to ensure that the monitor can at least come up and start to run, the above sections of the JUPITER system must be running in some reasonable fashion.

The console must report error conditions detected on the integrity check in some humanly engineered fashion to the console terminals if possible and to the indicators on the console board for the console tests themselves.

B.1.7 ERROR RECOVERY/RECORDING <TO BE IMPLEMENTED>

Error detection and correction in the rest of the system and in the console itself will depend upon the console software to become involved in the recovery procedures. The console must support the scan path diagnostic visibility/control scheme. The console software will be responsible for:

A). Performing required recovery procedures for the recoverable class of error. Depending on the nature of the fault this may require interaction with cpu firmware, hardware and operating system software.

B). Recording of recoverable error events in the system error data base (SPEAR) after recovery has been completed. Investigation of the data base will indicate when deferred

maintenance is required. The console will retain error messages on a stack until informed by the operating system that the error has been processed. In the case of power fail recovery the least recent error is retained in non-volitile storage for processing after recovery from the power fail has been completed.

**** list here the errors that require console intervention ****

B.1.8 ENVIRONMENTAL MONITORING <TO BE IMPLEMENTED>

The console software product will support the reliability goals by reporting marginal and failing environmental and power conditions in conjunction with the environmental monitoring module(EMM) . Decision control (where practical) will reside in the console for environmental conditions which fall outside parameters prespecified to the EMM's by the console.

The above conditions should ,in addition, cause the console software to pass information relative to the condition to the system error log(SPEAR) whenever it is possible to do so.

B.1.9 MEMORY RECONFIGURABILITY <TO BE IMPLEMENTED>

It must be possible to reconfigure the memory on the JUPITER from the console so that there is contiguous memory starting at physical address 0.

Reason: Currently BOOTS and the monitor require that there is contiguous good memory starting at physical address 0.

***** PROPOSED *****

B.1.10 PRIVILEGED COMMAND PROTECTION <PROPOSED>

There will be several commands implementable by an operator through KCRSP and KCCON Maintenance Console Software that are dangerous to the system if invoked inadvertantly. We would prefer that commands of this nature be safeguarded in some manner that tends to assure that the operator's action is intentional.

B.1.11 REDUNDANT BOOTSTRAP <PROPOSED>

Redundant files necessary to bootstrap the system and maintenance packs resident on both RL02 console media.

8.2 DIAG/MAINTENANCE SOFTWARE MAINTAINABILITY IMPLEMENTATION

The Diagnostic Engineering Maintenance software product is directed at the serviceability of the JUPITER product in the following areas:

1. Manufacturing build and test
2. Installation/acceptance
3. Preventative maintenance
4. Deferred maintenance
5. Corrective maintenance

A humanly engineered and well designed JUPITER diagnostic product will provide opportunities for cost savings and customer satisfaction at every stage in the product life. As the primary tool of the field service engineer the diagnostic product must support the maintenance strategies delineated in the Maintainability Engineering project plan (which later evolves into the maintenance plan for JUPITER). The primary goal of the maintenance software relative to the availability goals of the Jupiter system is to facilitate repair of the system in unscheduled remedial maintenance situations. The elements most responsible for a short repair that can be impacted by the maintenance software are:

1. Preventing loss of error information
2. Reducing trips to the site
3. Reducing the diagnostic cycle in the cases where on site diagnosis is necessary

***** IMPLEMENTED *****

8.2.1 HARDCORE <IMPLEMENTED>

Console Resident software should reside in non-volatile prom (hardcore), check out console module hardware functionality and provide a method to load additional routines from the console load device.

8.2.2 SCAN PATHS <IMPLEMENTED>

The maintenance software set must support the product strategy for diagnostic visibility and control into the rest of the system using the micro-diagnostics.

B.2.3 MAP OF NON-VOLATILE RAM <TO BE IMPLEMENTED>

At this time,a map of the contents and use of the non-volatile ram has not been specified fully. Currently, the Ram space is as follows:

RAM ADDRESS		FUNCTION
770000		NOT SPECIFIED
770626	!	ODT,POWER UP
	!	64 WORDS USED BY PROM
770776	!	INCLUDING:
		-BREAKPOINT
		-BAUD RATE
		-START ADDRESS OF LAST PROGRAM

Non-volatile RAM Contents

The 256 words of non-volatile RAM will be used to store information which must be maintained over powered down periods. This information currently includes:

1. The time-of-year of last power down.
2. The terminal characteristics for the console terminal lines.
3. The baud rates for 3 of the 6 console USART lines, excluding the video terminal line and the RS422 lines.
4. The KLINIK password.
5. The environmental conditions before power down (AC low) or whenever an abnormal environmental condition is detected.
6. System configuration information. This includes:
    1. The names and versions of CPU microcode to be loaded on autoload.
    2. which type of microcode to load into each I/O Port.
    3. which KC10 pre-boot program is to be loaded during auto reload - system or diagnostic preboot or none.
    4. The program name to be passed to the Boot program. This will be a limited character, complete filespec which is settable by command or by the Boot program itself.
    5. KC10 memory configuration, including Cache.
7. Default parameters, such as:
    1. the autoboot load device.

2. the Autoreload flag.

8. A record of the least recent error on the error stack in the console remaining in process in the event of a power fail during use of the monitor.

NOTE

This information will be backed-up on the console disk with a command for restoring this into the N-V Ram.

B.2.4 ISOLATION DIAGNOSTICS <TO BE IMPLEMENTED>

The isolation goals for the diagnostic set have been stated in chapter two and will be published in the maintenance plan. The ramp profile of Jupiter is such that if the console software and maintenance software are unable to isolate to the FRU on a solid fault, the system is probably down for some other reason than cpu cluster problems. In these instances it would be desirable to have the cluster verification as complete as possible before the engineer arrives.On detection of a solid fault the console software will be responsible for:

A). Gathering all available data pertinent to a failure and storing that data for error reporting purposes. In the case of power failure the least recent error should be available in non-volitile storage.

B). Notification to the console terminals of solid fault data in a humanly engineered fashion with considerations to customer sensitivities. The console, upon detection of solid fault conditions in the cpu cluster will attempt to localize the fault to the failing field replaceable unit and report the failing box,FRU, and nature of the problem to the console terminals and possibly a recommended course of action(call local field service office, assistance center after hours etc.)

C). Upon detection of a solid fault that cannot be localized to the FRU, the console will notify the console terminals in a humanly engineered fashion that system continuation is not possible at this point, that a self diagnosis session is in progress and possibly a recommended course of action as above. The console will then begin an integrity check of the cpu cluster via control files and the layered diagnostic hierarchy. The self-diagnosis control sequence should not jeopardize the integrity of the customers data.

B.2.5 FUNCTIONALS <TO BE IMPLEMENTED>

Functional tests are used to validate design completion at Design Verification Test time, Design Maturity Test time,Manufacturing FAI time, Customer Installation and Acceptance, ECO verification and for remedial troubleshooting and repair verification.

B.2.5.1 CPU <TO BE IMPLEMENTED> -

Functional tests should be designed to verify the Engineering Functional Specification of the KC10 mainframe. Some conversion work of the 22 functional tests is expected. In addition we are requesting additional functional testing specific to KC10. This included:

Pipeline

Worst case Instruction accessing

Worst case conflicts

Worst case Memory references

Algorithm set up for NEXT and SKIP target instructions (NEXTI and JUMPI)

MVE Forward Select Operations

AC Sets

Pager

Extended Addressing

FPA

B.2.5.2 EXERCISER <TO BE IMPLEMENTED> -

A set of the Functional tests should run under UETP and along with various Software Products be developed into a System Exerciser, to be run under TOPS20 operating System.

B.2.5.3 IO -

Functional testing of the I/O Box,Ports and Lincs is also required. We request a deviceless wrap around mode. An interactive test capable of verifying functionality of the mainframes and I/O interface.

A test is required to exercise all the functionality in an interactive mode (all busses and data paths active) at full KC10 speed.

B.2.5.4 ADAPTORS <TO BE IMPLEMENTED> -

A test is required to verify full functionality both internally and externally to include loopback support on the bus. This test will utilize system functional microcode.

B.2.5.5 DEVICE DIAGS <TO BE IMPLEMENTED> -

An extension to the above mentioned test will be a current device test adapted for JUPITER's architecture, or, in the future, a test specifically designed for new peripherals.

B.2.6 RL02 FILE UTILITY <TO BE IMPLEMENTED>

File transfer/conversion utility that allows:

A). Humanly engineered method for partial or full rebuild of RL02 diagnostic packs from Diagnostic released magtapes from stand-alone mode or from either tape or disk in user mode.

B). Humanly engineered method of patching and re-writing diagnostic files. The utility should accept indirect command files to facilitate automatic patching in a consistent manner.

B.2.7 TRACE UTILITY <TO BE IMPLEMENTED>

.P Snapshot utility that performs the functions previously associated with a trace utility, performs trace dumps, ucode verification and english text translations, time stamped configuration information and the like. Commands and parameters must be passable from indirect files.

B.2.8 RD PARTITIONING <TO BE IMPLEMENTED>

Partitioning of console firmware supporting remote diagnosis to support the console goal of making an RD kit salable as an option and removable for non-contract customers. i.e. a removable socketted prom.

B.2.9 KLINIK <TO BE IMPLEMENTED>

Traditional KLINIK exec/user mode functionality including the capability to pass information to/from other async lines on the console module. Implies character (in the clear) mode as the base level protocol. This version of klinik implementation should have as goals:

1. Minimize on-site human intervention required to run diags, gather error information and configure system

2. Minimize impact of future protocol changes and RD eco's

3. Maximize field services chances of isolating to the field replaceable unit remotely on the newer proposed peripheral set i.e. CI devices...this implies a RS422 line and contention arbitration protocol..it is recognized that there are project dependencies..

B.2.10 DN2X LINK <PROPOSED>

In the DN2x communications environment provide the functionality for the traditional dn2x backdoor link(the DN2x has no cty) and in addition some functionality for linking the backdoor of several dn2x's for diagnostic purposes in user and exec mode. This is a large problem that needs sizing. Objectives to be considered include:

1. Desirability of using existing 11 based diags for the subsystem

2. Remote diagnosis

3. Minimal impact to existing sub-system configuration guidelines, documentation, etc.

B.2.11 RECOGNITION <PROPOSED>

Recognition and prompting as in Tops20 style exec..numan engineering in this area can be a big time saver(labor) provided sufficient storage is available.

B.2.12 COURSE OF ACTION BUFFER <PROPOSED>

Provide a method for field service account representatives to tailor the course of action message suggested above for his site i.e provide a modifiable buffer in non-volatile storage that contains a message for the site person such as an operator in cases of solid hardware faults. The message could contain contract coverage hours, account rep names, phone numbers, after hours procedures etc.

digital

interoffice

memorandum

To: DISTRIBUTION LIST

Date: 26-Jan-81
From: Al Snyder
Dept: M.E.G. JUPITER
DTN: (8-)231-4747
LOC/MAIL STOP: MR1-1 S35

Subject: Fatal Memory Parity Errors In The IO BASE PAGE

At a recent CPU loose ends meeting the issue of movability of the IO BASE PAGE Register was a topic of discussion. The IO BASE PAGE Register is a hardware register that is held in the IOBOX and initialized by the CONSOLE at BOOT time. This register will point to a Page in Memory that the MONITOR will utilize to communicate to the IO Ports and the CONSOLE (see KCIO SPEC by Ted Hess for more detail). There was a consensus that at some point in time the Monitor may want/need to move the location of the IO BASE Page. In order to accomplish this the Monitor would need to change the register in the IOBOX. Since this operation would be done on a limited bases it was thought that having the Monitor communicate to the Console (via the IO BASE Page) and request the Console to change the register in the IOBOX would be adequate. This should reduce the Software coding effort and may also save some hardware. Regardless of what the final hardware/software resolution is there is still an issue of what happens when there is a fatal Double Bit Memory Error in the IO BASE PAGE. The purpose of this memo is to communicate MEG'S viewpoint on this issue.

We assume that in order for the Monitor to request the Console to change the IO BASE Register it will need to utilize the Communication region of the IO BASE PAGE. Since the Double Bit Error could have been in that area of the Page, this type of communication may not be possible. Additionally any IO interrupts (including APR level) may not be valid due to the error. For these reasons we have concluded that an acceptable approach for such error cases would be to halt with a FATAL IO BASE PAGE BUGHLT. Since the probability of getting a Double Bit Memory error in ECC correctable MOS, and having the error in that exact page is nil, a BUGHLT seems to be the best thing to do. The effort of writing code that could handle this error case and the unlikelihood of ever using it makes this a justifiable tradeoff. This does not mean that we will never need such code. Sometime in the future we may need to handle such an error if we find its occurrence is unexpectedly high. In making this decision we have

considered the following assumptions true.

- The Console will have visibility into the MBOX to gather error information (Address, syndrome, Data, etc.) for logging and FRU isolation.
- The Console will be able to report the fault.
- The Console will pass an error packet to TOPS when the system is reloaded.
- The Console will be able to reconfigure the memory and reassign the IO BASE Page if the problem is solid.
- The IO BASE PAGE does not reside in the CACHE (Double Bit errors are not detected in the Cache).
- The error is a result of a memory failure (to include the Bus) and not as a result of the IO writing bad data into the page. If the problem is the IOBOX or Port sending bad data, then the error should be flagged as such and the bad data blocked from memory.

This subject raises another question that needs to be addressed. If the error is in the Console Communication Region, how will the Monitor notify the Console of a BUGHLT? This is not just a problem under this scenario but also in the case where the IOBOX fails or goes away!

The final technical resolution on how the page will be moved may change. However unless the handling of such an error case would become intuitively easy our position will remain the same.

digital

interoffice memorandum

To: Judy Hall

Date: 16-Apr-81
From: Al Snyder
Dept: CSSE
DTN: (8-)231-4749
LOC/MAIL STOP: MR1-1 S35

Subject: ERROR CATEGORIES

Most of the attached document was part of a presentation I gave to the SPEAR group on Error Handling for the JUPITER. It occurred to me that this type of categorization may be what you are looking for. Rather than rewrite the document I merely added comments where I thought appropriate. Most of the document is motherhood but the Operating System section should give you some idea of the size of the task (I hope). The things I added to the document, per our MS discussion, are highlighted by (NOTE) or by (---) after an error. Is this a start on what you requested? If yes, what is the next step? If no, what is it you would like? I'll be glad to detail this more or do something different if you wish. I'll be glad to do whatever is necessary to assist you in assuring proper Error Handling on the JUPITER.

KC10 ERROR LOGGING METHODS

The following describes the Multi-level approach to Error Handling on the JUPITER system. Basically there is four levels of Error Recovery and Reporting.

- HARDWARE LEVEL ---
- CONSOLE LEVEL ---
- OPERATING SYSTEM LEVEL ---
- APPLICATION LEVEL ---

**** HARDWARE LEVEL ****

* The KC10 CPU has logic built into its design that DYNAMICALLY captures and PRIORITIZES error conditions. The HARDWARE then decodes the errors and produces a CODE that indicates the correct FIELD REPLACEABLE UNIT (FRU). This code is held in a register that is accessible by the CONSOLE no matter what the state of the system (up or down).

* The MICROCODE will then force the hardware to RETRY the operation without intervention from the Operating System or the Maintenance Console.

* If the retry is successful i.e. the error was TRANSIENT, the MICROCODE will continue with system operation with no adverse effect on the USERS or the MONITOR.

* The following are some examples of errors that can be detected and recovered from in this manner.

NOTE

IF THE MICROS CANNOT RECOVER FROM THESE ERRORS THEN I DON'T THINK THERE IS MUCH THE MONITOR CAN DO. IN FACT IF THE MICROS CAN'T RECOVER THEN IT IS DOUBTFUL THE MONITOR WILL REMAIN ALIVE. IN SOME CASES THE MONITOR MAY BE ABLE TO BUGHLT AND SAVE SOMETHING??? SOME OF THESE ERRORS ARE CURRENTLY HANDLED AT MONITOR LEVEL AND WE PROBABLY WANT TO CLOSELY VISIT THAT CODE... ...SEE BELOW.

O VBUF ERR	O PF OR TAG IN ERR	O IDL E1E2 ERR
O E1E2 ERR	O LRF OR STR ERR	O EC ERR
O INX ERR	O INSN 09-17 ERR	O OP2X ERR
O EA ALU ERR	O IDL EA ALU ERR	O IDL OP2 ERR
O SCA IN ERR	O INTERNAL SCA ERR	O X,Y ERR 0,1
O X,Y ERR 2,3	O VCODE ERR	

O DIR 0 PAR ERR --- See PCDPE O DIR 1 PAR ERR --- See PCDPE

O DIR 2 PAR ERR --- See PCDPE O DIR 3 PAR ERR --- See PCDPE

O SBME ERR --- This equates to SBE (Single Bit Error) on the KL. This error will be handled and reported by the KC10 firmware/hardware and the Console. TGHA and the MF20 hardware handled this in cooperation with Monitor. A lot of the extensive code that is there for this error can be greatly reduced or eliminated. We may want SPEAR to do some error prediction and prevention for this error i.e notice a threshold of SBMEs has been reached and ask the Monitor to Purge Memory. The Monitor would need to support this with a Special Job function (JOB 0 or a newly assigned special job).

O CWDBE ERR --- This error was not detected on the KL10. If bad data was read out of cache on a writeback it was detected at the MBs and the ERA was held to show Cache Writeback. This error will be handled by the Micros.

O PCDPE ERR --- This equates to Cache Directory parity error on the KL. "FATCDP Bughlts and CD0 Stopcodes" This code should be reviewed since this error will be handled by the Micros. If the Micros cannot recover then TOPS will not be able to do much.

O MOS ADR PAR ERR --- This error may require some changes from the way the Operating System currently works. Currently address parity errors are handled totally by

the MONITOR. BUGHLT FATAPE if fatal address parity error. I think the MBOX micros will be able to retry the reference and in some percentage of the time recover. If the Micro recovers then TOPS probably will not be told of the error. If the Micro cannot recover by retrying the reference then the Operating system will probably not be able to do anything more to recover. ???

**** CONSOLE LEVEL ****

* The Maintenance Console provides Dynamic Error Recovery for many classes of error in the KC10.

* There are error conditions that the hardware/firmware detects that it itself cannot recover from. When an error is detected the hardware will notify the console of the fault and freeze the state of the machine.

* The console can snapshot the state of the machine, find the error source, correct the bad bit (or reload correct data), and continue the machine operation at the point where it was frozen. The FRU is held in a hardware register and by the Console.

* The following are some examples of errors that are recovered from in this manner.

NOTE

THESE ERRORS ARE FATAL. IF THE MICROS CANNOT RECOVER THE MONITOR WILL NOT BE ALIVE TO EVEN DO A JRST 4,, .

O IPUT ERR	O ISET ERR	O MAC ERR 0,1
O ESE ERR	O MAC ERR 2,3	O CRA ERR
O ED ERR	O FPA UCODE ERR	O CU UCODE ERR
O DMU UCODE ERR		

* The Maintenance Console has the ability to dynamically snapshot error information for all errors types detected in the KC10 cpu.

* In cases where the hardware/firmware recovers automatically the Console can retrieve enough error information from the passive latches to correctly identify the FRU.

* In the cases where the hardware requires intervention from the Console to recover the Console will be able to isolate the FRU and in many cases will be able to isolate the failing network or chip.

* The Console has the ability to disable Automatic Retry and force the machine to halt when any error is detected. It can then take an entire snapshot of the machine.

* In all cases the Console will assemble an error packet with the isolation information contained and send it to the error file for later analysis or reporting by SPEAR.

* The Console can also print all or a portion of the error information on the HARDCOPY or VT100.

**** OPERATING SYSTEM LEVEL ****

* The Operation System provides the top level of support for error recovery.

* Severl classes of errors will require the Operating System to intervene when they occur.

* The Operating System has the ability to retry many operations and in some cases when the operation is fatal it can abort that process without crashing the entire system.

* The following are some examples of errors that will require operating system intervention to recover from.

  - DBME ERR --- This error equates to a double bit error in the MF20. However, unlike the MF20 there is no hardware or software (TGHA) that allows spare bit substitution. Many of the ideas used for the KL as far as handling of a Memory error can be followed. However, I think we should look at this code carefully and make appropriate changes i.e. I see no need for a core scan, Error logging will be totally changed (no ERA), and most of all we should make sure we look at each cycle type and write appropriate code to handle each case. What I am refering to is; A DBME on a IBOX read will have to be handled differently that a DBME on a IO read, etc.

  - TB ERR --- This equates to Page Table Parity Error in the KL10. All Pager Refills are done by the EBOX i.e. KL style paging on the current 20s. Much of the error code to handle this fault should be able to be replicated with only minor updates and corrections.

  - MBUS ERR --- This error equates somewhat to a channel memory parity error. However, it is a new error e.g. we did not check parity on the CBUS of the KL10 and that is what this effectively does. I think we will need quite a bit of new code to handle this. However, a considerable amout of this code should be able to be common for other errors detected in the MBOX and IOBOX.

  - MEM DATA ERR --- This error equates somewhat to a MB PAR ERROR in the KL. However, it is not the catch-all for every data path like the MBs were in the KL. Much of the code should be able to be common with DBME.

- ARRAY BUS ERR --- This error equates with SBUS error (KL10, DMA, TOPS10 only) on the KL. However, it has only one meaning; a parity error was detected on the Bus between the MBOX and MOS, unlike the KL where it can have several meanings. Some of the code should be able to be common with DBME. We should look closely at this error because it also is cycle dependent.

- MDL DATA ERR --- This error equates to a MB PAR ERR in the KL (only when the EBOX of the KL was requesting Memory Data. It usually will mean a parity error in the data cache and will be corrected automatically. It could be the result of a mux failure. This error will probably require some new code. The closest thing in the KL would be an AR or ARX parity error on data read from cache and that code just doesn't apply to the KC10.

- MDR DATA ERR --- Same as above.

- VMA ADR ERR --- This error was not detected on the KL10. It will require new code.

- NXM ERRORS --- A NXM on the JUPITER is completely different from what is was on the KL10. Traditionally a NXM is the result of the Memory not responding with an address acknowledge pulse when an address is sent out on the bus. In the JUPITER; NXM is a MBOX function only. The MBOX compares the PMA (that it PLANS to address MOS with) with a RAM of valid address ranges. If the address is valid the MBOX will use it (after some translation) to address the MOS. If it is not the MBOX will set NXM. We should look at this error and see if any of the old code applies. I think the probability of recovery from this error is higher than in the past (I assuming a transient error in a CKECL chip rather than a MG10, MH10, MB20 not responding).

- MOST IO ERRORS --- These errors will be handled much like they were in the past. However, there will be many new errors and we will require new and more code to address them. This is the area I think we could get into some trouble if we are not careful. I have heard many rumors on why the RH10 to RH20 conversion was so difficult and I certainly can tell you the problems we saw in the field. I think talking to Tony (and anyone else that knows what went on) and getting the straight scoop would be beneficial. We should be very careful in this area and look at all the code very closely. I have more thoughts here and would like to discuss them verbally... I don't know how many IO errors we will be required to handle but I do know it will be many more than in the past.

- COMM --- As above...

o ALL TRADITIONAL PERIPHERAL ERRORS --- As above...

* The Operating System will generated an error packet for any error that it is involved in.

* If the system crashes the Operating System will generate an entry with appropriate reload information.

* The Operating System will totally support the error reporting of packets presented to it by the Console.

* The Operating System should cooperate with spear to provide dynamic error logging and dynamic error prevention. Dynamic error prevention may be possible for errors such as single bit mos errors (sbme). When some error threshold is reached for this error spear may request the monitor to purge memory. This will reduce the probability of a double bit mos error (DBME) occurring.

* The operating system will provide a software crash dump (image file of memory) when an error occurs. There will be an optional switch that will allow this dump to occur on recoverable errors (bugchks).

* All the required tools to analyze these dumps should be supported by the monitor. Documentation on Crash Analysis should be developed and supported. We want to include this in SYSTEM level field service training.

**** APPLICATION LEVEL ****

* Spear will be the top level tool for error analysis.

* Spear will not be utilized for error recovery. It may be utilized to provide some level of error prevention as explained in the operating system level.

* With the advent of the lower levels of error recovery and reporting, spear will not be required to do extensive error analysis for a large portion of the errors.

* Spear will be required to do error analysis and reporting for the more traditional errors (especially io). Basically all errors that require monitor intervention will require some level of spear analysis.

* Basic bit to text will be required for all errors.

* Error trend analysis will be beneficial for all level of errors.

* SPEAR may be required to do some degradation analysis if a high number of recoverable errors are occurring.

* Spear will ??????

+---------------------------+
! ! ! ! ! ! ! ! ! interoffice
! d ! i ! g ! i ! t ! a ! l !
! ! ! ! ! ! ! ! ! memorandum
+---------------------------+

To: ERROR HANDLING DIST. LIST

Date: 13-Jul-81
From: AL SNYDER
Dept: CSSE (JUPITER)
DTN: (8-)231-4749
LOC/MAIL STOP: MR1-1 S35

Subject: TOPS-20 ERROR RECOVERY FOR KC10 ERRORS

1.0 INTRODUCTION

I had been working on an Error Spec for MBUS errors detected by the MBOX and after finishing the Hardware section I tried to move on and document what the Monitor would/should do when these errors occur. I quickly realized that I had no idea what we would do for Monitor level error recovery under the new IO scheme. I am particularly concerned with CPU errors on IO references that were traditionally recovered from by the Monitor; however, the same thought should go into errors on CPU references and Port/Adapter errors on IO references.

I would like to hold a meeting in the not-to-distant future to discuss this. The following is some preliminary information that may help set the forum for the meeting.

2.0 NEW IO SCHEME

Under the old Massbus configurations most errors (CPU or IO Controller originated) had to be retried by the CPU. The Controller would detect the error, store status and error information, and interrupt the CPU. The Monitor was responsible for determining which controller was failing and what to do to recover from the error. In most cases the Monitor would perform some number of retries and pretty much control all aspects of the recovery. The Monitor would be notified, regardless whether the error originated in the CPU or the Controller. If the error was a CPU error the Monitor would receive an APR interrupt and the IO would be notified of the fault by a Channel Error that would eventually produce an IO interrupt e.g. MB PAR ERR.

With the new NI/CI configurations most errors can be retried by the Intelligent Controller (PORT). This holds true for CPU and IO errors. The addition of this new intelligence will undoubtedly require change to the current Monitor scheme of error recovery. One example of a potential change may be as follows:

0 MBOX MBUS ERROR -

For the sake of making this an easy example let's assume an IO Read data cycle is in progress. IO Read is; data being read from memory - passed through the MBOX - through the IOBOX - through the Port - over the CI - and onto a disk, tape, or other CI link.



A parity error has been detected in the MBOX at location (P). Briefly; the MBOX will send the bad data to the IOBOX along with a FAULT Code indicating the error. The IOBOX will complement the Parity and send bad data with good parity to the Port. The IOBOX will pass a FAULT Code to the Port. The appropriate Port will respond (via TAG match) and retry the reference. The error information captured by the MBOX will be logged by the Console and the error information captured by the Port will be available if desired. More detail on this error is available in the MBUS Error Handling Spec.

As you can see, unlike the Massbus, the Monitor need not be involved in this error recovery. Note that there was no CPU interrupt i.e. APR interrupt generated. This keeps it rather clean as long as the Port can recover. The question; what if the Port can not recover? It seems to me that having the Monitor perform additional retries would be useless. What can the Monitor do? Can it reset that Port and start over? Will that do any good if the error originated at the CPU? Must the Monitor do things differently depending on the FAULT CODE e.g. if the error is a result of a solid CPU fault, then all Ports will be affected and no IO operations can be continued. On the other hand, if the error was a result of a solid Port fault, then that port could be reset and the other Ports allowed to continue.

I certainly have not addressed all the problems and questions, nor any of the solutions, that arise with this new scheme; however, I hope this will give us a starting point.

In order to give some perspective to the magnitude of this issue, I have listed some errors that I believe fall into this class. They are listed in order of potential occurrence (my guess).

- o DOUBLE BIT MOS ERROR -

  This error indicates two data bits are failing on a word read from Mos Memory. It can occur on any cycle type.

- o MBUS TRANSCEIVER PARITY ERROR -

  This is the above error. It indicates an error has been detected on data being transfered between the MBOX and the IOBOX. It can occur only on IO cycles; Reads, Writes, or Command Address.

- o ARRAY BUS PARITY ERROR -

  This indicates an error has been detected on data being transfered between Mos Memory and the MBOX. It can occur on any cycle type.

- o IO ADDRESS LATCH PARITY ERROR -

  This indicates an error has been detected at the register in the MBOX that holds the Command Address data. It can only occur on IO Command Address cycles.

- o NXM -

  This error indicates that an invalid match has occurs on the Address Ram in the MBOX. The Address Ram holds the lower and upper address boundaries of the available memory. It is loaded by the Console. This type of NXM is slightly different from the traditional NXM in the KL; MA/MB, MG/MH, or MF memories (no ADR ACK). It can occur on any cycle.

- o BNK PARITY ERROR -

  This indicates a parity error has been detected in the Bank Control Ram in the MBOX. This error can occur on any cycle.

- o MBOX DATA PATH ERROR -

  This indicates an error has occurred in the MBOX data paths (MDP module). This error can occur on any cycle except IO Command Address.

These are just some of the error conditions that we may expect to see that will require Monitor intervention if the Port is unsuccessful at retrying the operation. There are more errors that come into play for CPU references i.e. Translation Buffer, E/I BOX errors etc. We can discuss these later. I believe this will be

enough to get us started.

I would like to set up a meeting to discuss this after you've had a chance to review it and form your own opinions. Please give me a call if you have any comments or would like to see any additional

+---------------------------+
! ! ! ! ! ! ! ! ! interoffice
! d ! i ! g ! i ! t ! a ! l !
! ! ! ! ! ! ! ! ! memorandum
+---------------------------+

TO: ERROR DIST LIST

CC: Max Weinfuss
Nick Howgate

Date: 5-Aug-81
From: Al Snyder
Dept: CSSE (JUPITER)
Ext: 8-231-4749
LOC/MAIL STOP: MR1-1 S35

Subject: SPEAR HANDLING OF DOUBLE BIT MOS ERRORS

We held a meeting to define the SPEAR requirements for analyzing and reporting DBME (DOUBLE BIT MEMORY ERRORS). In doing so, we also defined some guidelines for future errors. The following is the minutes of that meeting.

1.0 DMBE HANDLING

When a DBME occurs the MBOX hardware/firmware will capture enough information to successfully isolate the FRU (see DBME SPEC). This information is held, in passive registers, for the Console to retreive. The Console will retreive this information even it the system crashes from the Memory Error.

The Console will build an Error Packet and pass the information to the Error File, via the Monitor, for use by SPEAR. Since the information in the Error Packet contains all the information required to isolate the FRU, SPEAR will normally not be required do any analysis. Spear will be required to do BIT-TO-TEXT translation and report the error.

1.1 MONITOR ENTRY TO ERROR FILE

The one case where SPEAR could do analysis is as follows: If the error occurred on a Memory read, for a requested word, the Monitor will trap (PAGE FAIL) with an error. The Monitor will collect information about the fault. Specifically, the Monitor will receive a Page Fail Word containing the VMA, a Hard Page Fail Code, whether the reference was User or Exec, and possibly the PMA. Whether the Monitor crashes or a User gets a hard error is

dependent on the type of data.

It is desirable to have SPEAR correlate any crashes, User or Monitor, with the Error entry from the Console. The Console entry may be time stamped and logged long before or long after the Monitor or User crash. As such, the Monitor should provide an entry into the error file that would allow SPEAR to correlate these two error entries. An example of useful information would be the PFW, the PMA, and a time stamp. Any additional information that would assist SPEAR in correlating these errors could be reported. However, the goal should be to keep it as simple as possible for the Monitor.

If the Monitor BUGHLTS or BUGCHKS, then the normal BUGXXX and reload information should be provided.

2.0 ADDITIONAL THOUGHTS ON SPEAR

Besides discussing DBME we talked briefly on several other SPEAR points. The following is some initial thoughts on those points. Some of them will, undoubtedly, spark additional meetings.

1. ERROR FILE FORMAT -

The SPEAR group (referred to from now on as SPEAR) needs to understand the format and logging procedure for the Error File. Will it be the same as the TOPS on the KL? Preferable so! Will the Console be making entries without going through the Monitor (Jim J. suggestion at one time)? If so they should be identical to the Monitor entries. SPEAR suggested that it is best to keep the Monitor code, for error logging, as simple, clean, and easy as possible.

This topic will require a follow-on meeting with Software Engineering.

2. CONSOLE ENTRIES -

SPEAR must have the Console entries in Binary.

Any entries from KCTRAK, RSP Dump or the equivalent, should be contained in one contiguous Event entry and be in binary.

If the Console disables some portion of error logging, due to an excessive rate of recoverable errors, it should notify Spear. Spear should be able to request the Console to turn that error logging feature back on if so desirable.

SPEAR will hold a meeting with Diagnostic Engineering to better define some of the above.

3. GENERAL DISCUSSION POINTS -

We discussed several general topics that will need to be addressed in order for Spear to support the JUPITER.

o DYNAMIC ANALYSIS -

We (CSSE) believe Spear will be required to run dynamically and report certain error thresholds, performance degradation (due to errors), and possible configuration changes, such as Memory, Cache, etc. An internal meeting has been scheduled to discuss this. Memo, minutes, and additional external meetings to follow.

o EXPANDED DECIDER PROCESS -

It may be required to have system job or Spear, running dynamically, to report errors and configuration changes on the Interconnect Busses; especially in High Availability configurations. Spear would find out that a PORT or HSC50 is getting an enormous amount of recoverable errors. It may "decide" that system performance (IO Bandwidth) has decreased significantly and request the Monitor to switch to another PORT/HSC50 that was dual ported to the disk structure. (Example only).

Meetings and Memos on this to follow.

o SPEAR AND MANUFACTURING -

As stated earlier, we believe Spear will not be required to do extensive analysis to isolate faults (FRUs) in the CPU Cluster. However, it may be possible for Spear to collect information from the Console and Monitor, and with some incremental effort, provide Logic Network or Chip isolation for Manufacturing. This is not currently planned, nor has any meetings been scheduled to discuss it. It may be a great financial benefit to Manufacturing and Field Service to have such support.

The JUPITER has the ability to trap and log intermittent failures. Thus, when some high number of intermittents occur on a given FRU, the Field Engineer will replace the module. Unlike the KL, he will feel very confident the module was truely failing and indeed will fix his machine. However, he will not know what chip is failing. When the module is returned to Manufacturing for repair, neither will the Manufacturing Tech. Since the problem was intermittent it may not be reproducible. Now what? Do we put it back in stock? If so and it returns two more times, without being repaired, current policy is to scrap it. Dollars down the drain! If Spear could provide Logic Network or Chip isolation we could possibly save some money.

This may not be required, nor a viable solution, if the UBON and IC tester can isolate these type of failures. It's a thought we may want to bounce around. Certainly not a FCS project!

o ENVIRONMENTAL INFORMATION FOR HARD ERRORS -

We talked about Spear correlating hard failures to Environmental faults or rapid changes. SPEAR thought it may be beneficial to have an Environmental entry from the EMM/Console when a hard error occurs. This would be useful if Spear would have some way to determine a change from a previous environmental state. This topic will be looked at in more detail later.

DIGITAL

INTEROFFICE MEMORANDUM

TO: Distribution list

DATE: 18-AUG-81
FROM: Al Snyder
DEPT: CSSE (JUPITER)
DTN: 231-4749
LOC: MR1-1/S35

SUBJECT: MBOX ISSUES MEETING

The following is the minutes of the MBOX issues meeting held on 11-AUG-81.

1. CACHEABLE PAGE, LOOK, AND LOAD BITS -

   These bits are not required and will be dropped from the KC10 Architecture.

   The Cacheable bit allowed a User Mode, Functional, Memory Diagnostic to run in CORE and not trash the Cache. Currently, no such diagnostic is planned. Deleting the Cacheable bit may preclude writting such a diagnostic in the future. However, CSSE does not require such a diagnostic for Field Service use and Diagnostic Engineering (J. Kirchoff, D. Tongel) agreed with deleting the bit.

   The Cache Look and Load bits provided instruction control for writing into and reading from Cache (see KC10 Specification). The Monitor Group does not require the Look and Load bits. They will be removed from the KC10 Architecture.

2. CACHE DISABLE -

   The KC10 has a feature that permits dynamically disabling of Cache when certain errors are detected (see MBOX Error Specs). It is a RAMP requirement to have the Cache disable occur without Monitor or human intervention. Additionally, for most error cases the disable will occur without crashing the system (Single Bit Errors and Cache Directory Errors). Under current hardware/firmware implementation any ONE quadrant or ALL FOUR quadrants can be disabled e.g. Cache quadrant two is failing with Single Bit errors and is dynamically

disabled. The system is now running degraded with three Cache quadrants (6K). Cache quadrant four fails with a Single Bit Error. The entire Cache (all four quadrants) is disabled.

This is acceptable with CSSE and meets the RAMP requirements, as-long-as the disabling occurs dynamically, for both error cases, without crashing the system.

3. DOUBLE BIT ERRORS ON NON REQUESTED WORDS -

Under current hardware/firmware implementation when an error is detected in a NON-REQUESTED word, on a MOS Read, the MBOX will send a Page Fault Code back to the IBOX for the REQUESTED words (assume two word IBOX instruction fetch... ..see DBME Spec for details). Thus, when the EBOX calls up that instruction it will get a Hard Page Fail even though the data (REQUESTED WORD) was not in error. This would reduce system availability.

One solution would be to add hardware and firmware that would allow the MBOX to know which words, of the quad-word fetch, were requested by the IBOX. The Mbox would only send a Page Fault Code back when one of the requested word was in error. This is the optimum solution but would require module real-estate for logic.

A second, and more viable, solution is to leave the MBOX hardware/firmware as-is and solve the problem in the Ebox Page Fault Handler (Microcode). When the MBOX detects a DBME it will block the Cache write enable(s) for the faulty word(s) but write the good words into Cache. The MBOX will send a fault code back for the REQUESTED word(s) even if the error was in a NON-REQUESTED word. The Page Fault Code sent to the IBOX would indicated a DBME occurred on that quad word reference. When the EBOX calls the location up for execution it will Page Fail. The EBOX Page Fault Handler will recognize the DBME Page Fault Code and retry the reference (assume one word refetch). This time the requested word would come from Cache and no error would be detected. The refetch/retry would be successful and no Monitor level Trap or crash would occur. If the original error was in the requested word; the refetch would go to MOS and another Page Fault and subsequent Monitor Trap would occur.

The only time this solution fails is; if the MBOX detects a DBME and the Cache has been previously disabled. Then the refetch/retry will Page Fail even if the error is in a NON-REQUESTED word. However, in order have this error case occur; the Cache would have had to been disabled, due to an error, and then a DBME occur in MOS. I think that is a pretty low probability. CSSE will accept this solution if we can't free up chip space for the ideal solution.

digital

interoffice memorandum

To: Don Hooper E85
Judy Hall L10
Ron Mclean E85
Pat Sullivan E85
Dave Tongel E68
Mike Uhler E85

Date: 5-Feb-82
From: Al Snyder
Dept: CSSE (JUPITER)
DTN: (8-)231-4749
LOC/MAIL STOP: MR1-2 S35

Subject: EBOX STORES AND HARD PAGE FAILS

I have scheduled the PERIPHERALS conference room next Friday (12-FEB) at 4:00pm to discuss handling of Hardware Page Faults during Ebox Stores.

IN REVIEW:

I BELIEVE THIS IS THE SITUATION!

Ebox stores that resulted from instructions; running in fast code i.e. MOVEM, and/or have been previously Write Tested do not wait for MRESP after the request has been sent out. The Ebox sends the Request, waits for MGRANT and then proceeds the next operation. The Ebox assumes the request will not Page Fail.

THE PROBLEM:

This works fine under normal conditions, however, a serious problem surfaces if a hardware fault occurs while the Mbox is finishing the Store. Upon detection of the error the Mbox will send a hardware Page Fail back to the Ebox. In the mean time the Ebox could have executed several more instructions. Subsequently, the returning hardware Page Fail will be extraneous to the current operation.

THINGS CAN GET EVEN WORSE:

The Ebox could have overwritten the Source AC of the original Store. The number of instructions the Ebox may have executed is dependent on; the instruction mix, any further Mbox request that it may have sent, and how long, from the time the request was

honored, until the Mbox sent back the Page Fault. How long after the Mbox honored the request until it sends back the request is dependent on; where the error was detected, the type of Store (Physical or Virtual), whether a Cache Hit occurred, and whether a Writeback had to be done.

The only current (assumed) solution is to bring the system to an immediate, Clock Stop, HALT! I believe this is error situation will prove to be a critical hole in the reliability of the Jupiter.

I'd like to discuss the above, at the meeting, and hopefully come up with a viable solution.

digital

interoffice memorandum

To: DIST

Date: 12-Feb-82
From: Al Snyder
Dept: CSSE (JUPITER)
DTN: (8-)231-4749
LOC/MAIL STOP: MR1-1 S35

Subject: MOS MEMORY CONFIGURATIONS

While discussing Memory Error Handling with the SPEAR group I realized there was some confusion over how the Console configures memory and what it must do to reconfigure the memory. I'd like to take a minute and try and clear up this confusion.

NOTE

PORTIONS OF THE FOLLOWING WERE EXTRACTED FROM THE MBOX SPECIFICATION AND OTHER STATEMENTS ARE BASED ON MY KNOWLEDGE OF THE WHAT MUST HAPPEN TO CONFIGURE AND RECONFIGURE MEMORY. I HAVE VERIFIED BOTH WITH THE DESIGN ENGINEER. ADDITIONALY I HAVE MADE SOME GENERIC ASSUMPTIONS ON WHAT MUST BE DONE TO MAINTAIN CONFIGURATION INFORMATION FOR PROPER OPERATION OF THE SYSTEM AND FOR SERVICEABILITY.

1.0 INITIAL CONFIGURATION

Each time the machine powers up, the console must go thru an initialization routine to determine which banks contain memory modules, what size the arrays are, and ensure that bank 0 contains modules. The console must force the Mbox to address each memory bank and record the number and types of arrays present.

The Mbox contains a Mapping Ram that allows bits 11-16 of a memory address to be mapped to one of the available banks. This Mapping Ram allows the 64K or 256K Ram boards (pairs) to be plugged into any slot. The only restriction is the slot zero constraint as mentioned above. This Mapping Ram also provides the Console with the ability to select or deselect the desired Array

configuration for system operation.

2.0 RECONFIGURATION

The same basic steps must be accomplished if, for whatever reason; hardware errors, software problems, add-ons, etc. one wishes to reconfigure the memory. The SYSTEM must be halted or put to sleep (depending on system sleep implementation) and the Console must reconfigure the Memory. There are no provisions in the hardware to allow any functional PDP10 program (including the Monitor) to reconfigure memory. However, the Console can programably deselect any of the memory banks with the exclusion of physical bank zero. If bank zero is faulty then one must replace it or move it to another bank. You must insert a good module set back into bank zero.

3.0 SERVICE STRATEGY

We will assure Field Service takes full advantage of the ECC features of the MOS Memory i.e. only replace the array boards when a double bit error occurs or when SPEAR (FAULT MANAGER) indicates that a predetermined threshold of single bit errors has occurred. I should note; the Fault Manager will actively monitor the arrays for single bit errors and will (after a threshold is reached and via a TBD interface) request the Monitor to purge memory. This should eliminate some of the SBEs and greatly reduce the probability of a DBE. This means Field Service should seldom replace an array do to faulty MOS Rams.

However, there will still be cases where an array is failing and the customer does not want to keep his system down while the service engineer is in route. The solution is to Halt or, if possible, Sleep the system and have the Console reconfigure the memory. Additionally, SPEAR may determine the SBE threshold has been reached and request the operator to reconfigure memory deselecting the degraded array thus preventing an uncontrolled future crash.

Ideally we would have a mechanism that would allow SPEAR, the Monitor, and the Console to cooperate and do the selection dynamically. This would only happen after confirmation from the operator.

4.0 CONFIGURATION FILE

Because we do have a requirement to be able to reconfigure the memory, the console will have to maintain a configuration file on the current selected array information. This file should be kept in NV Ram or on the RL to prevent destruction on power down

or reload. This will allow the system to be reloaded using the previous configuration file and thus eliminating the possibility of reusing faulty arrays.

There should be a mechanism to allow the operator or at least the service engineer to force a desired configuration. Additionally, there should be a command to allow cold start initialization of the memory.

Since the Arrays are not sterilized the only way SPEAR can determine faulty array boards is by physical addresses. This requires SPEAR to maintain configuration information on the current selected and deselected arrays. SPEAR must also know which slots the given physical address is selecting. Since, the Console will have this information, in its configuration file, it seems only sensible that it would pass the information to SPEAR or at least make the file accessible. If the Monitor requires this information then the file should be available to it also.

SPEAR will also have the knowledge of when and which array boards were replaced or swapped by the service engineer. This knowledge will be gained through the NOTIFY command to SPEAR. The Field Service Engineer must understand the criticality of using this NOTIFY command. If SPEAR does not know of module swaps, changes, etc. then it will undoubtly be requesting unwarrented memory reconfigurations.

5.0 SUMMARY

I hope this clears up some of the misconceptions on how the hardware/firmware configures memory. However, I have probably opened new avenues of ambiguity. We must obviously get together and spec this out in detail. I fully expect the answers to be in the repair portion of the error spec and the RSP error spec.

digital

INTEROFFICE MEMORANDUM

TO: Distribution List

DATE: 8-March-79
FROM: John Shebell JS
DEPT: M.E.G.
EXT: #6405
LOC/MAIL STOP: MR1-1/S35

SUBJECT: Attached

The attached article was written by Bill Demmer sometime during 1974 while he was on a consulting assignment. It is a good picture of the current trends in computer systems maintenance and provides a lot of insight into IBM's strategies in the RAMP area. Since my copy was very poor I had it re-typed. I hope it is of interest.

dmo
attachment

TRENDS IN COMPUTER SYSTEMS MAINTENANCE

The importance of Maintenance has been receiving gradually increasing recognition at the top management level of computer systems manufacturers. This long-term trend has been accelerated within the last five years for some manufacturers, including IBM, by expensive, unanticipated, maintenance problems, and IBM, because of its dominance, tends to set the trends for the computer industry. In addition, there is good reason to believe that IBM's continuing dominance is due, in no small way, to the widely-acknowledged high quality of IBM's service to the user, of which maintenance is a significant part.

Three facets to the increasing recognition of the importance of maintenance are evident.

1. Maintenance activities are given their own organization within most companies, with lines of authority frequently extending to a "vice president for field engineering" or equivalent at the highest level of management. Where maintenance and sales share common management in other companies, the level at which a pure maintenance organization reports to a "sales and service" manager has been rising.

2. Concern that the maintenance operations make an identifiable contribution to a company's earnings is rising. Increased attention is being given to cost accounting rules so that this contribution can be meaningfully measured.

3. Maintenance is being given increasing recognition as a technology in itself. With this increased recognition, the maintenance organization now is making major contributions to the design of new systems, starting at the early stages of product planning and continuing through hardware and software specification and design.

Developments in computer systems maintenance technology include some new techniques for minimizing the cost of maintenance and increasing reliability and availability; several older techniques are being extended to new areas, or developed and applied in greater detail. The following techniques are gaining increased acceptance:

.use of data communications for remote maintenance

.configuring computer systems with more redundancy to decrease the possibility that a single hard failure will shut down the system

.hardware error detection

.logging of the electronic environment when errors occur for use in scheduling maintenance and for aid in diagnosis

.automatic recovery from errors using

- -hardware error correction
- -command retry (peripherals)
- -instruction retry (central processing unit)
- -recovery routines in the operating system
- -automatic reconfiguration when an error proves permanent.

.use of service processors for maintenance functions and the operator interface

.designing out high failure rate components such as switches, indicating lights, and connectors

.programmed fault detection and isolation using

- -diagnostics of several types
- -"cookbook" maintenance procedures
- -program-driven module testers (at repair centers)

.computerized management of spares inventories

Much of the emphasis in the development of maintenance technology is based on an increased appreciation of the importance of intermittent failures, particularly in digital circuits; intermittents are more prominent at least partly because failure rates have been falling.

Some older maintenance techniques are being used less, especially on the newer, commercial computer systems. These include techniques based upon a detailed understanding of the function of computer equipment, and requiring use of oscilloscopes, meters, test points, extender cards, indicating lights, or special purpose test equipment. Margin tests are no longer used very much for electronic circuitry. Margin tests are no longer used very much for electronic circuitry--it is now believed that they introduce stresses which tend to increase the failure rate.

Trends affecting the maintenance organization are:

.increasing user acceptance of "mixed" maintenance arrangements, i.e., those involving more than one maintenance organization at the same facility

.increasing cost of spares inventories, at least partly because of the trend toward larger, more specialized circuit modules

.decrease in skills required at the lowest level of maintenance because of improvements in programmed fault detection and isolation, and because of the trend toward remote maintenance

.more use of operators in fault diagnosis

.more repair activity at repair centers or depots rather than at the user's site

.more use and improved orangization of computerized maintenance data bases

Remote Maintenance

Perhaps the most obvious specific trend is that toward remote maintenance. Most computer manufacturers use some form of remote maintenance, interpreting the term broadly. Here, however, we mean remote maintenance involving a telecommunication link which is established specifically for a maintenance function. The following are the major categories:

1. A data link is established from a remote terminal to the failing computer; a technical expert operates the remote terminal. In some cases the remote terminal may assume most of the functions of the local operator's console. This is particularly appropriate for software problems.

2. A data link is established from a local terminal to a remote computer; no connection is made to the failing equipment. The field engineer or specialist searches data bases on the remote computer for material related to the problem at hand.

3. A data link is established from the failing system to a remote maintenance computer. Testing procedures are transmitted to, and results are transmitted from, the failing system. One or more terminals will be connected at various points for interfaces to the field engineer and maintenance specialists.

All three categories are already in use by one or more of the major manufacturers.

IBM attempts to provide all backup levels of maintenance support to the on-site Customer Engineer (CE) with remote maintenace facilities. Support specialists are sent on-site only for exceptional problems or to placate an irate customer. On the other hand, the lowest level of maintenance is almost always conducted on-site; it is the on-site CE who calls in the remote support facilities. In many cases the problem at hand may be resolved as the CE describes the problem to the specialist; if not, at the Field Technical Support Center (FTSC) the specialist

has many resources available, including complete sets of field manuals, logic diagrams, program listings and other maintenance publiscations; mockups of computer control panels to aid in visualizing the CE's observations and actions; the computer-based facilities of RETAIN (Remote Technical Assistance and INformation Network); and the communications facilities and know-how to call in higher-level backup teams extending to the engineering design or programming group responsible for the particular system in trouble.

RETAIN/370 also permits extension of the data link from the FTSC directly into the larger 370 systems (145 and up). With the data link established, the RETAIN computer is able to transmit special diagnostic programs to the failing system, the specialist at the FTSC is able to exercise control from his terminal, and results of tests are displayed both to the specialist and the on-site CE.

Remote maintenance aids are especially appropriate to computer models which exist only in relatively sparse population, since concentration of failures to a relatively small group of maintainers will help build higher levels of skill. But all computer models start their product lives at sparse population levels, and this is when failure frequency is highest and skill level lowest. In spite of the fact that the RETAIN/370 data link is not available to the 370/115, 125 and 135, we think that remote maintenance aids connecting to the failing system are applicable to medium scale computers and even down into minicomputers.

IBM is not alone, and perhaps not even the leader technically, in providing remote maintenance. All manufacturers use verbal telephone communications when the on-site field engineer is having difficulty, and telecommunications equipment is a "natural" for at least some level of remote maintenance. Five of the other major manufacturers have aspects of remote maintenance in operation or under development and the others can be expected to follow.

COMPUTER MAINTENANCE STRATEGY

In most early computing systems from the manufacturers point of view the considerations involved with the reliability and maintenance of computer systems have traditionally been secondary items to cost and performance. If customer satisfaction was not lost due to reliability or maintainability weaknesses then the manufacturer was content. For the most part this is still true today, except that the requirements to keep the customer satisifed have changed significantly and also the profit motive has entered the maintenance business. The customer can only commit himself and his business to greater dependence on the computer system if it meets his expectation as much in the parameters involving reliability as it does in the cost/performance criteria. However, with the state of the art today in measuring and defining reliability criteria it is a much more subjective and emotional consideration than cost/performance. The latter point has caused a dilemma to the P & L motivated designer and manufacturer. It is very difficult to measure the return on the investment and one must generally be content with a measure that indicates whether it was acceptable on the whole or not. The problem of assigning values to individual elements of a total Maintenance Plan (which includes much more than the hardware reliability features) is guesswork at best as far as their impact on customer satisfaction is concerned. With the rise of the profit motive in the area of maintaining computer systems more work has been done on being able to define costs associated with the various activities involved. Therefore, the manufacturer is able to assign some values to the constituent parts of his Maintenance Plan where the functions are there to influence servicing costs. Thus, in the system or unit design the manufacturer can make some supported tradeoffs, but because most elements in a Maintenance Plan effect customer satisfaction also even these tradeoffs tend to be quite subjective. The ideal situation for the designer is to have a funtion that will aid customer satisfaction, but which can be fully justified by reductions in service cost.

When initiating the system design phase of any computer product program, the Maintenance Plan for that product must also be considered. This is not just a list of desirable reliability functions that should be designed in the hardware, but it must encompass all system associated aspects as well, such as software requirements/contributions, operator and CE training, operational environment of the product, maintenance tools, etc. The goal of this Plan is to,

1. Reduce the number of interruptions that might occur in the product, and

2. to minimize the impact that interruptions might have on the system operation.

Fortunately this goal is generally not in conflict with the maintainers objective of keeping servicing costs as low as possible. Reliability is the term often used to indicte how frequent an interruption occurs, Serviceability deals with how long it takes to recover from an interruption, and the word Availability to indicate some form of total impact of both. Hence, the term RAS has become much used in this area of computer systems. Keeping the system or product operational (Availability) has become the primary objective of most Maintenance Plans via the two step plan inidcated above.

PREVENTING INTERRUPTIONS

The system designer will attempt to incorporate in his design ways and means of preventing interruptions. These generally rely on hardware and software approaches with some occasional human interaction being planned for. One of the first things to be addressed to aid in the prevention of interruptions is the tradeoff of cost and failure rate of the individual components being used in the design. In most comerical systems this is an economic tradeoff and not a technical limitation, although it is often implied to be such. From the design of the organization of the system the use of highly reliable components throughout is a most straightforward way of reducing interruptions.

The next step is that of recognizing when a component does fail at the time it fails. This recognition may help subsequent isolation of the failing components, but by itself it does nothing to prevent an interruption and can be the vehicle to signal an interruption. However, it can also be the vehicle to signal for an attempt by the system to recover from the failure. The mechanisms to achieve recovery are varied and include many hardware, firmware, and software techniques. A partial list of the significant hardware and firmware mechanisms used on System 370 follows.

1. The main memories and control storages all utilize an Error Correction Code (ECC) which detects and corrects all single bit failures in the memory plus provides detection of many multiple bit failures. While this is a relatively expensive function in logic and memory cost and it adds at least logic delays in usually critical timing areas, it probably has been most worthwhile as it does prevent what otherwise would be in the majority of situations a total system interruption.

2. Instruction Retry is a mechanism usually under the control of a microprogram which will prevent an interruption when it can successfully redo an operation that had failed on a previous attempt. It should be noted that under conditions of a permanent failure, instruction retry is not useful. Retry capabilities can be instituted in many different

places within the system such as the channel and control units although in these areas only a limited number of functions can be retried.

3. The hardware will also record a great deal of information on the status of the unit at the time a failure is recognized. This status information can then be made available to the system level software which can attempt recovery from the failure.

4. Significant system interlocks and protection mechanisms have been incorporated into System 370 architecture to be sure that an erroneous act by an application or user program will not cause a disruption to the system or any other program being run.

In the Operating Systems used on 370 a variety of software functions are incorporated in order to recover from a failure and prevent an interruption. The Recovery Management System (RMS) includes routines for trying to get around or minimize the impact of machine and channel failures. It can call upon error recovery programs for each I/O subsystem which will perform a retry of the operation or an analysis to see if what is failing in bypassable. Where hardware is available it will seek alternate paths to the desired I/O device, and may even call upon the operator to try mounting the recording media on another unit. Thus, with adequate redundancy at the unit level (whether it be a tape drive or main memory,) the operating system can reconfigure and perhaps degrade the system, but nevertheless keep it in useful operation until a human decision is reached about what course of action is appropriate.

MINIMIZING IMPACT OF INTERRUPTIONS

Once an interruption does become unavoidable there are many things that can be included in the Maintenance Plan to attempt to not only keep the duration of the interruption small but to also assure ease of start up. These bring the customer and the service organization into the picture along with the hardware and software system.

It was mentioned earlier that upon recognition of a failing component, it was often possible for the system hardware or software to devise a way of continuing system operation, although at some expense such as lower performance, less configuration flexibility or increased reliability risk. Therefore, it is desirable to replace or fix the failing component at a convenient point in time. This situation is usually referred to as deferred maintenance and is the most desirable form of maintenance with the exception of preventive maintenance. Deferred maintenance is subject to a degree of scheduling that in service labor cost alone can significantly reduce the cost of the service call over that of an unscheduled interruption. The system hardware or software can provide further significant aid in this situation to

minimizing the impact of having to identify the failing component. At the time the malfunction is detected, the system is able to record a variety of status information that can then be made available for analysis. With the state of the art today very little of this type of analysis can be handled by the use of some form of automatic analyzer, such as a software program running as a problem program in the system. However, in many cases the recorded information alone is ample data to identify the faulty component. Programs are then available to edit this data for ease of interpretation to the individual responsible for calling the customer engineer (CE) or the CE himself. Some of the System 370 recording and formating functions are:

(a) CPU Hardware Logout - Status bits within the Central Processor Unit are stored at defined locations within main memory. An extended logout field is provided to handle model dependent information and the greater number of status bits for the larger system.

(b) Control Unit Sense Bytes - Each individual control unit records its status bits and upon issurance of a Sense Command from the CPU will pass these in the form of Sense Bytes to main memory.

(c) Recovery Management System - This or an equivalent software function can record such information as the number of correctable errors occurred in memory, the number of successful retries, damage to a timer function, etc. This along with job identification parameters and time of day data are then written in the system error recording data set (SYS 1 LOGREC).

(d) The Statistical Data Recorder (SDR) and Outboard Recorder (OBR) - These routines write into the SYS 1 LOGREC a record of the temporary I/O device plus all channel type errors.

(e) Environment Recording, Edit, and Print (EREP) - This is a utility program that will edit and print selected contents of the data stored on SYS 1 LOGREC.

The programs identified above are those available specifically with the OS/370 operating system, but are representative of what is generally available for the recording and formating of error information.

There are situations in which it is not possible to continue system operation and deferring maintenance until a later point in time. However, in such instances it is still often possible for the various hardware recording functions to operate and usually even some of the software recording activity can be carried out. For example, if a CPU error is uncorrectable and RMS can determine there has been no damage to the control program, then it may be possible to only abort the one program and continue the

processing of others. However, if there is damage to the control program or if it cannot be determined, then the system must stop and call for manual intervention. However, this is not done until a logout is completed and is recorded. This situation is often called a hard or exigent machine error which terminates system processing.

Even under these conditions there are aids that the software system coupled with user planning can bring to bear to minimize the effort of restarting. The automatic restart facility can be used to have the programs that were terminated to automatically be rescheduled without the formalities of a new job submission. Furthermore, with the savings of work queues provided by the warm start facilities and with the appropriate checkpointing by the user, complete rerunning of the entire job can often be avoided.

The entire above discussion on minimizing the impact of an interruption centers around the recognition of an error and the recording and presenting of information about the error to a level of detail that would permit the customer engineer to diagnose which component has failed. The instantaneous capturing of machine status after a failure is probably the optimum method known today of localizing transient or intermittent failures. Since data on system reliability for early System 360 installations indicted a preponderance of these intermittent failures, the design of System 370 hardware and software strongly reflects the emphasis placed on recording and recovery.

This technique as a fault isolation tool is one of the most promising because of its ability to be useful in situations where an error cannot be forcibly recreated. By having the hardware store sufficient status information including that of the unit's control latches it should be possible to analyze the data and identify the failing component. Today it is the analysis activity which is the limiting factor in this technique. A simplistic approach to this might be to compare the actual recorded data bits against bit patterns previously generated by assuming a failure of a given component. However, it is easy to see that the number of previously generated bit patterns would have to be overwhelming, and some very unique methods of segmenting the two sets of data must be found. I believe the development of algorithms for performing this type of analysis will be underway within the next few years and that this will provide a major step not only in failure isolation techniques but also in automatic system recovery operations. Due to the dependency on the smallest details of the design implementation, it should be noted that the analysis parameters would be of little value across machine families or on units only functionally equivalent.

Since the above methods are not yet perfected, one of the major tools currently in use to minimize the impact of an interruption by reducing the time to repair the unit is the diagnostic test. The traditional diagnostic program was generally an after-the-

fact program written using only the same instruction set as was available to any programmer. This meant that a set of complete functional tests could be generated (again, to be complete the number would be extremely large) and therefore most permanent errors could be detected. However, the problem of translating a functional failure definition into a physical hardware element was often impossible and additional hardware aids were needed. These aids could be in the form of hardware checking circuitry, instructions for interrogating other control or data latches generally not available to the programmer, and even methods for the setting of data and control latches. The last item was recognized as a requirement shortly after checking circuits were introduced as it was necessary to find a way to verify that the checking logic was working correctly.

These techniques were only slowly improving the actual diagnosis capability of the diagnostic program. It was the advent of the use of the control store, and more specifically the writeable control store (WCS) that gave the diagnostician a very powerful new tool for fault isolation. Because the microprogram can work at a very detail level usually governing the setting of control and data latches on a machine cycle by machine cycle basis, it is a far simpler (but sill difficult) task to generate a set of tests that can isolate to a very low level of logical content, which is then readily transferred to physical location. As more and more logical functions are being housed on the smallest replaceable physical unit, the task of identifying that replaceable component becomes easier. With the cost of hardware continuing to be reduced and with the cost of service continuing to increase, the use of highly functional packaging (despite the increase in spare parts cost due to lower multiple part usage) will tend to become more predominant. Thus, the only major drawback to the microdiagnostic is the fact that it requires recreating the failure to be effective and therefore is generally only useful where there is a permanent failure. In cases where there is marginal design or component, the use of microdiagnostics under a veriation of voltage or clock frequency might help recreate the intermittent fault. Errors resulting from data dependencies or multiple unit interactions have a very low probability of being found via microdiagnostics.

Since System 370s that have a writable control store use semiconductor memory as the technology, an external device is required to house the microprograms due to the volatility of the semiconductor memory. The external storage device used is a floppy disk, which is also used for the housing of microdiagnostics. Generally, any loading of the normal microcode for instruction processing from the floppy disk into the control store is preceeded by an automatic CPU checkout microdiagnostic. When microdiagnostics are to be loaded they are usually found on one or more separate floppy disks and should be used in a prescribed sequence. In order to reduce the size of the hardcore (that circuitry which must function to be able to use microdiagnostics) to the minimum, some of the initial

microdiagnostics will be executed directly from the floppy disk rather than the control store. Then the technique of advancing to other tests using only logic that has been previously checked expands the scope of the microdiagnostics to test to entire processor or I/O system. Where microdiagnostics are used in this manner today they usually require a dedicated unit and therefore, are used in an off line mode.

For system 370 there have been developed for most I/O devices a set of on line diagnostic tests (OLTs) that can be run while normal customer processing is proceeding. These OLTs interface into the operating system via an executive program (OLTEP) that is actually part of the OS or DOS operating system. Since these tests run as a problem program and must be transmitted across the IBM standard I/O interface similar to any functional I/O commands, it is difficult for them to be effective as a fault isolation tool. However, they do serve as good repair verification tools and also for unit verification following installation of engineering changes.

To get around the situation of having the standard I/O interface in the path, many I/O control units have a maintenance panel capability that allows the customer engineer to exercise the subsystem at a more detailed level which permits better fault isolation. Furthermore, there is the capability to use the control unit and one I/O device while that same control unit and its other I/O devices continue normal customer processing. The utilization of this in-line technique and also the on-line OLTs require considerable diligence on the part of the customer engineer. This is because he is operating switches and controls on the console or other control unit panels which are at least physically close to those which can disrupt the system operations.

With the limited introduction of the service processor in the System 370 Model 125, a broader concept of its potential is not difficult to visualize. The 125 service processor would appear to provide no new function, except that it stores all the system error recording data, which is stored on operating system data sets (SYS 1 LOGREC in OS) in earlier 370 models. It loads the micro-programs used in the various sub processors, operates the console, and handles the error logging under control of its own microprogram. Under an expanded concept of the service processor there could be a new cable between it and the CPU or control units, where the service processor could run a form of microdiagnostics to locate problems in the other subsystem. Although the hazards from accidently interfacing with the systems operation could be significantly reduced compared to the previously mentioned human involvement, the problem of providing an adequate microdiagnostic type test simultaneously with system operations is a formidable one in concept. Therefore, the use of the service processor may be somewhat limited to a monitoring and recording one until some auomatic-analysis of recorded data might be accomplished by it. The elimination of duplicate hardware in

many control units might foster some increased use of service processors to do off line testing.

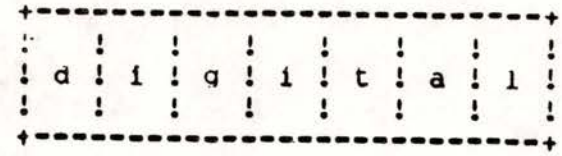
Once analysis capability arrives the question will change to individual system service processors versus one large remote multiple system service processor. The concept of performing fault isolation from a remote source is being offered today with RETAIN 370. This is a remote maintenance service available to IBM customers engineers and requires that each using system have a communications adapter attached to transmit data to a central/regional site. Data transmission can be under the control of the On Line Test Executive Program, and it is hoped that by duplicating the information presented to the local customer engineer a regional expert at the remote location can provide enough guidance to significantly reduce long service calls. Today in practice RETAIN is not often used and when it is, it is primarily as a two way conversation. In addition to the expertise, RETAIN central maintains data on engineering changes, frequent troubles and corrections and other historical information that might be of use to the maintenance task. I believe it is the plan someday to be able to perform some level of anaylsis on the data transmitted from the system by a computer at the remote site. Probably only IBM can afford to experiment with such a system, but eventual benefits to service costs look very significant. Thus, while the potential of remote maintenance appears very significant, it will be some time before the majority of maintainers can justify something more than the telephone call back to the experts at some central site.

To briefly summarize the major trends in the world of maintenance I would list the following.

1. Emphasis on automatic recovery via redundancy, correction, and retry.
2. Full recording of complete machine status at time of failure.
3. Use of microdiagnostic techniques for off line fault isolation facilitated by larger functional replaceable units.
4. The use of separate intelligence (probably remote) to aid maintenance activities.

In the future I would expect to see these more throughly integrated into a total system plan. The degree to which they are present in System 370 today is quite significant since very little thought was given to them at the time the basic System 360 architecture was developed.

GARY CLEARWATER	MR1-1/S35
RICK ELLISON	MR1-1/M2
BOB MALONEY	MR1-1/S35
DAVE ROCKWELL	MR1-1/S35
DON WERTH	MR1-1/S35
NIGEL WEBB	MR1-1/S35
JUDY HALL	MR1-2/L10
MIKE UHLER	MR1-2/E85
PAT SULLIVAN	MR1-2/E85
JACK ROSEN	MR1-2/E68
SCOTT ROBINSON	MR1-2/E85
JOHN MURRAY	MR1-2/E85
DON HOOPER	MR1-2/E85
GLENN GARVEY	MR1-2/E68
ARJUN KALRA	MR1-2/E85



interoffice

memorandum

To: DISTRIBUTION

Date: 21-Feb-83
From: STU KENNEDY
Dept: CSSE-JUPITER
DTN: (8-)231-4749
LOC/MAIL STOP: MR01-1/S35

Subject: COMPETIVITE RAMP INFO

In an attempt to better define the RAMP requirments for the next 36 bit LCG machine I have collected some competive RAMP data on the machines the Jupiter will be competing against in the markplace of the 80's. First is a list of some of the RAMP features on other vendors systems.

1.0 IBM SYSTEM 38

- o Instruction retry capability.

2.0 IBM 303X SYSTEM

- o Machine check recovery by instruction retry.

3.0 IBM 4300 SYSTEM

- o Instruction retry via checkpointing and retry arrays.
- o User process termination on unretryable instructions/errors.

- o FRU isolation for intermittent or solid errors.

4.0 IBM 3081 SYSTEM

- o Full ECC checking on main storage. Single bit correction, double bit detection.
- o Dual CPU processer with capibility to remove a failed CPU from configuration while processing contuines on other processer.
- o Processer fault checkpoint retry. Retry up to 7 times.
- o 90% hardware error detection. Direct FRU isolation in intermittent faults.
- o 35000 scan bits on LSSD.
- o Built in diagnostics.
- o Full remote diagnosis and support.

5.0 NAS ADVANCED SYSTEMS AS/7000/9000

- o Instruction retry on CPU fault condition.
- o Command and Channel retry.
- o Remote support/diagnoise facility.
- o Up to 9 K Bytes of status information written to floppy on each CPU or channel error.

6.0 AMADHL 470/570 SYSTEMS

- o Auto retry of instructions due to machine check conditions.
- o Extensive parity checking in I and E units.
- o ECC checking on main and buffer storage.
- o Remote support/diagnosis facility.

- o History rams for audit trail of system activities.

7.0 SPERRY UNIVAC 1100/60 SYSTEM

- o 100% fault detection in cpu data path.
- o Parity checking on storage elements and logic redundancy on arithmetic and control circuity.
- o ECC checking on main storage. Error correction on control storage.
- o Dynamic FRU isolation for solid and intermittent faults.
- o Fault recovery via macro instruction retry with saved data elements.
- o Fault recovery via program transplant to other processe.
- o Built in logic analyzer.

Attached are 3 technical reports on the RAMP features of 3 competive systems. The first is an overview on the Univac 1100/60, the second is a description of fault handeling on the IBM 4341 and the third is a series of articles on the IBM 3081 series.

AVAILABILITY, RELIABILITY, AND MAINTAINABILITY ASPECTS OF THE SPERRY UNIVAC 1100/60

L. A. Boone, H. L. Liebergot, and R. M. Sedmak

Sperry Univac, Blue Bell, Pa. U.S.A.

ABSTRACT

This paper describes the fault tolerant capabilities of the SPERRY UNIVAC 1100/60 Information Processing System, a recently announced medium scale general purpose computer system. In the 1100/60, a variety of techniques is employed for the detection/correction and isolation of, and recovery from, most single-bit hardware faults as well as many multiple-bit faults. An approach for checking fault detection circuits is implemented using a comprehensive fault injection system. A four level maintenance philosophy based around the built-in fault handling logic, scan network, and intelligent support processor and console provides for rapid location and repair of the failing logic.

During development, substantial resources were devoted to assure the quality of the 1100/60 design, recognizing that fault tolerance strongly complements, but is not a substitute for, design quality and correctness. An evaluation of the various fault handling features was carried out to provide measures of system availability, reliability, and maintainability.

I. INTRODUCTION

At Sperry Univac, the acronym ARM stands for Availability, Reliability, and Maintainability. ARM concepts include the organizational procedures used to develop systems; the tools and techniques used during design, development, and manufacturing, and the logic, firmware, and software that are included to minimize the effects of a failure. The latter ARM features include many of the fault tolerant features developed over the years.

Several important trends in the computer industry have strongly influenced decisions with regard to ARM at Sperry Univac. The evolution of computer applications from primarily a batch processing orientation to a stronger emphasis on demand and real time processing has resulted in greater dependence on computers and thus in increased sensitivity to their availability and reliability.

The second major trend relates to hardware technology. The increase in gate densities in LSI and VLSI chips has effectively brought the cost of hardware down, to the extent that it is becoming more practical and cost effective to incorporate fault tolerance in the computer. [1]

A third trend lies in the area of field maintenance. The costs of labor for maintenance activities is increasing at a rate close to that of inflation. In addition, as systems become more complex, it becomes increasingly more difficult for customer engineers to master the detailed workings of the entire system.

The reaction to these industry trends is to de-emphasize manual field diagnosis procedures and to stress the incorporation of built-in automatic fault detection, correction and recovery capabilities. The SPERRY UNIVAC 1100/60 Information Processing System reflects this increased emphasis on fault tolerance in commercial computers.

II. ARM PHILOSOPHY FOR 1100/60

A. ARM In Previous SPERRY UNIVAC 1100 Series Systems

The SPERRY UNIVAC 1108 [2,3], introduced in 1965, was the first SPERRY UNIVAC 1100 Series computer to offer both a multiprogramming operating system and multiprocessing configurations. These two capabilities reflect a general ARM approach that has been carried forward in other Sperry Univac systems - the attempt to isolate a problem to either a particular job in the system or to a particular unit in the configuration.

The success of this approach is determined largely by the error detection attributes of the system. For a typical 1108 system, error detection consists of parity in the main storage and processor general registers. For the successor 1100/10 and 1100/20 systems, this coverage was enhanced to include parity on the I/O channels and in some of the mass storage control units, while the main storage utilized an error detection/correction code instead of parity. Maintenance on the central complex (instruction processor, input/output unit, and main storage) is performed using a built-in maintenance panel and diagnostic programs.

The 1100/40 system has all of the previous error detection capabilities, and in addition, a maintenance controller as an adjunct to the maintenance panel. The controller incorporates a scan compare capability that allows operations in the processor to be examined after each clock cycle and compared to known correct data from a magnetic tape. Any difference can be used to indicate the location of the problem area. This was the first automation of the maintenance task for 1100 series systems.

In the 1100/80, additional error detection was provided for the input/output unit, and a cache memory with parity was added. The maintenance controller was replaced with a maintenance processor - an intelligent unit that can write several of the registers in the central complex and read almost all of them. The customer engineer's interface to the system is via CRT rather than a maintenance panel, and the maintenance processor can function even if most of the central complex is disabled.

B. ARM in the 1100/60 - General Approach

A fundamental requirement for the 1100/60 was to produce a system whose ARM attributes matched the processing modes of the future with the technology of today. The price/performance goals allowed the selection of proven ECL technology for the instruction processor (IP) and cache, TTL for the input/output unit, and 16K MOS chips for the main storage. In addition, the IP is microprogrammed and uses four-bit-slice microprocessors to achieve a reduction in component count. Established design rules allow adequate temperature, voltage, and timing margins. The basic unit processor and most expansion features are packaged in one cabinet and air-cooled. A block diagram of the 100/60 system is shown in Figure 1.

1604-8/80/0000-0003$00.75 © 1980 IEEE

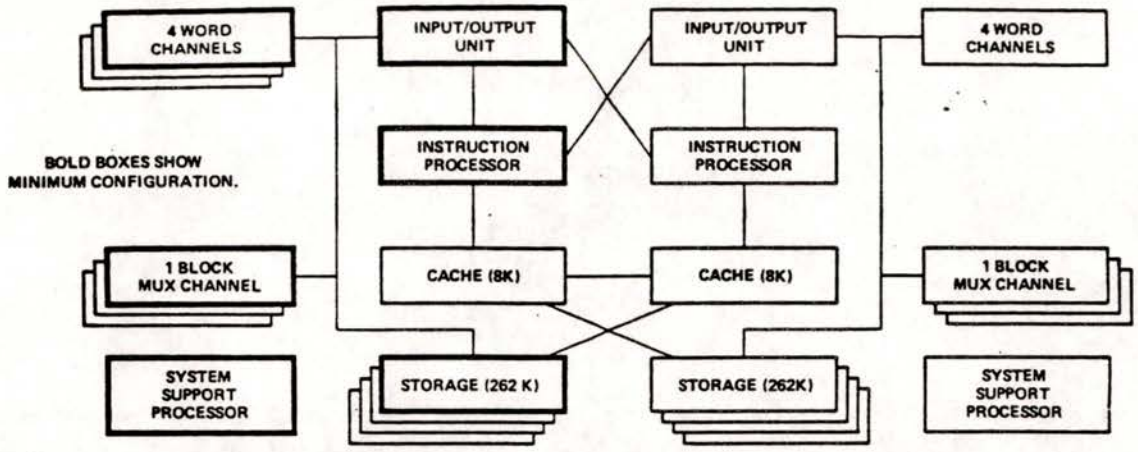


Figure 1. System Block Diagram (two processor, two I/O configuration)

Error detection has been given increased emphasis in the 1100/60 relative to previous systems. This provides protection from incorrect results, aids in system recovery and reconfiguration, and helps to isolate a failure to a replaceable unit. The 1100/60 uses duplication, coding techniques, and parity to provide error detection throughout the system.

The main storage and the IP's microcode control storage have error correction capabilities to allow error recovery to be transparent to the user. The cache memory architecture allows recovery from an error by automatically disabling an area of the cache and retrieving desired data from the main storage. Other solid faults are handled by reconfiguration. Extensive retry capabilities are provided to recover from transient or intermittent faults in the central complex.

III. DETAILED ARM IMPLEMENTATION

A. System Characteristics

The 1100/60 instruction processor is microprogrammed and is based on a high usage of LSI microprocessors. The amount of hardware required for the IP has been kept low by implementing a large portion of the control functions in microcode. The net effect of such an approach is to replace the traditional mass of SSI/MSI gates with high-density LSI arithmetic and storage components. The high throughput of the 1100/60 is achieved by the use of multiple microprocessors, overlap at the microinstruction level, prefetch of instructions and operands, and other design techniques [4].

Figure 2 is a block diagram of the main data paths of the IP. Each 1100/60 macroinstruction and interrupt is performed by executing a series of microinstructions. The execution of each microinstruction consists of bringing data from the general register set, main storage, or other source through the shifter into the subprocessors. Here, using multiple microprocessor chips, an arithmetic or logical operation is performed, combining the data from the shifter with data from the local storages or the accumulators, and placing the results in the accumulators internal to the microprocessor chips. At this point, the data may be placed onto the main data bus and written into local storage, the general register set, or main storage. These data movements are controlled by a microinstruction that partitions the work between microprocessors, selects the various resources (registers, local storage, etc.) to be used, and initiates selection of the next microinstruction.

The 1100/60 central complex contains in addition to the IP, an input/output unit (IOU), an optional cache or storage interface unit (SIU), main storage unit (MSU), and a system support processor (SSP). The functions of each of these units are well known except for the SSP, a freestanding, intelligent processor that employs a CRT and keyboard and serves as the system maintenance and operator consoles. In addition, this unit carries out the function of system partitioning, control storage loading, fault injection, and some assorted system support tasks. The system software and the IP microcode and hardware communicate with the SSP through interrupt mechanisms.

B. Fault Detection

The philosophy in the 1100/60 is to detect 100 percent of all single-bit faults in the data path, and many single faults in control logic. The rationale for such a philosophy is based on the principle of minimizing the probability that a user's data could be corrupted without the system detecting or correcting the erroneous operation and signaling the operator. As a general rule, faults are detected in storage elements by the use of parity codes, while redundancy is used for arithmetic and control circuitry. The various detection circuits are strategically placed in an effort to achieve a high coverage in hardware such as storages, which have the highest failure rate, and in areas such as the main data path, which has a high usage and where a large impact on the system might be experienced if a failure occurred.

An overview of the fault detection techniques used in various portions of the 1100/60 IP is given in Figure 2. The general register set, local storages, and the shifter input selector employ conventional parity; the shifter and, as discussed previously, the subprocessors (including the main data bus) are duplicated and compared. In addition, parity checks are used on control storage, while duplication is used on the control storage address generation and logic function generation circuitry.

An example of fault detection through redundancy is shown in Figure 3. Each of the two 36-bit master subprocessors is paired with a duplicate subprocessor that performs the same function on the same data as the master subprocessor. During each microcycle, only one of the master subprocessors can drive the main data bus; and when one of them is chosen to drive it, its duplicate drives a duplicate bus. At the end of the cycle, a comparison is made between the data on the two buses; any discrepancy will cause the operation to be interrupted.

During the development of the 1100/60, an observation was made that is contrary to most claims about fault detection and its associated performance impact. Many sections of the data path are duplicated and compared to achieve fault detection. However, the duplicated logic serves another purpose: it provides additional output drive capability (i.e., loading) for the functional circuits. For example, if a functional circuit has 8 unit loads on its outputs, the addition of a duplicate circuit could reduce that number to 4 by splitting the loads between

thus can not use dupl

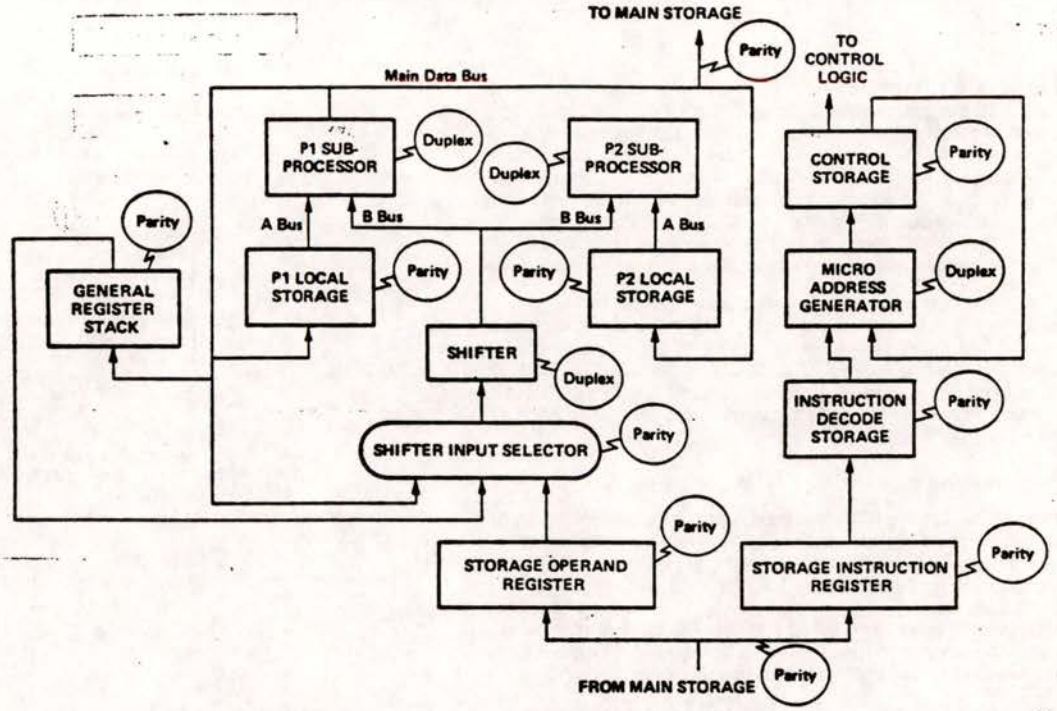


Figure 2. Block Diagram of the 1100/60 Microexecution Section, Showing Sample Applications of Fault Detection Techniques

the functional and duplicate circuits. This allows a reduction in the propagation delay time of the output stage of the functional circuit. In fact, as additional portions of the data path were duplicated to detect faults, it was discovered that the combined effect could be an increase in performance compared to a strictly simplex design. This result points out the fact that fault detection need not have an adverse effect on performance.

Several factors have contributed to keeping costs low for the fault detection features:

- Design approach of incorporating fault detection mostly in high-usage and high-failure-rate sections of the logic.
- Incorporation of most of the control functions in microcode, which is stored in LSI storage components where fault detection is very economical.
- Heavy use of microprocessors, which are well-ordered, bus-oriented logic structures that lend themselves to conventional fault detection techniques.
- The philosophy of designing fault detection circuitry at the same time as functional logic, and designing functional logic that lends itself to fault detection.

As a result of these, and other factors, the extensive fault detection mechanisms require about 15 percent of the total CPU logic. This translates to less than 15 percent of the manufacturing costs attributable to the detection logic.

C. Error Correction

Error correction techniques are applied in two primary areas: the main storage unit (MSU) and the IP's control storage. The MSU employs an error correction code to correct single-bit faults and detect double-bit faults in the storage array chips during every read reference. When a double-bit fault occurs, an error signal is sent to the requesting unit indicating that the data should not be used.

Error correction is also utilized in the microprocessors' control storage, although a different approach is taken than in the main storage. When a single-bit fault occurs, the parity code stored with each microinstruction will permit the detection of a fault. If a fault is detected, a macroinstruction retry is attempted. Should the retry fail, an interrupt then allows a correction procedure to be initiated by the system support processor. The procedure involves rewriting the failed portion of control storage. After each attempt at rewriting, the SSP will read the data from the failed portion of the control storage to verify the proper correction. When proper correction is achieved, the SSP signals the IP to restart execution.

D. Fault Isolation

There are two major techniques used for fault isolation in the 1100/60. One technique relies on the high level of coverage provided by the fault detection capabilities in the processor. For example, referring to Figure 2, it can be seen that because of the application of the duplication and comparison technique to both the shifter and the two subprocessors, it is possible to isolate a failure in that portion of the microexecution section down to one of those three logic sections.

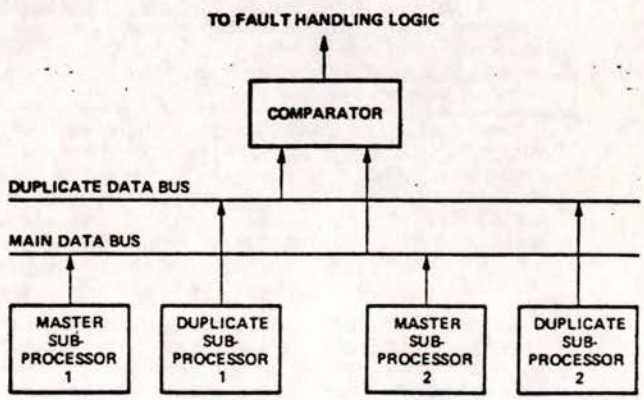


Figure 3. Example of Fault Detection through Duplication in the 1100/60 IP

The other major technique employed in the 1100/60 for fault isolation is the diagnostic program, a tool used after symptoms of the failure have surfaced. This approach is usually employed when the failure has not been isolated sufficiently by the fault detection logic. Diagnostics are constructed in the form of microroutines or macroroutines, which are run on the failing unit. Frequently, these diagnostic programs make use of a tool in the 1100/60 known as scan compare [5,6] which is a method of determining the states of major test points in the IP.

Utilizing the two methods, the capability exists to isolate automatically (i.e., without the need for manual diagnosis) any failure in the main data path to one or two printed-circuit cards. The obvious values of such a feature are to reduce the time to repair the central complex (and hence reduce field support costs) and dramatically improve the availability of the system to the user.

E. Error Recovery

The basic requirement in the development of an integrated error recovery procedure is that the computer system must deal with both solid and transient faults. When a solid fault occurs, any operation affected will continue to produce incorrect results. Detection and isolation of such a fault is relatively easy, but recovery from it without manual intervention and repair is frequently difficult unless some type of error correction or masking capability exists in the system.

In contrast, a transient fault is more difficult to detect and isolate due to its lack of persistence during diagnosis. However, recovery from transient faults in many areas can frequently be achieved in a more economical way than for solid faults. To accomplish this, the main items required are the ability to detect the error close to the time of occurrence, a mechanism to stop processing upon error detection, and the capability to reset the operation to a valid state which existed prior to the error, utilizing information that was saved during execution.

The error recovery procedure in the 1100/60 IP is designed to deal with both solid and transient faults. An overall picture of the recovery procedure is shown in Figure 4. The procedure uses a combination of hardware and firmware to implement fault detection, abort the operation in progress, analyze the fault, and initiate retry. This approach allows the recovery procedure to utilize the same microexecution resources as the macroinstruction set utilizes, resulting in a cost savings and a sharing of fault detection capabilities.

When a fault is detected by any of the detection circuitry discussed in section B, microexecution is halted via hardware by forcing the control store to execute continuously a no-operation microinstruction and by blocking the loading of internal registers and storages. Execution is suspended until a hardware timer expires, during which time no execution or change of machine state occurs. This allows time for transient failures to die out. The period of the timer may be adjusted for different program environments.

When the period expires, execution restarts in a fault recovery microroutine, which analyzes the failure and assembles a fault status word to be presented to the software. Then a check is made to determine whether the failing macroinstruction can be retried. If no retry is possible, macroinstruction execution is restarted and an error interrupt is presented to the operating software to allow the failure to be logged and software recovery of the failing system to be attempted.

If the failure is retryable, another procedure occurs. The 1100/60 IP incorporates a "save" storage for machine state restoration. The save storage is a small memory element which captures general register set operands and addresses during GRS reads. The fault recovery microcode can restore original operands to the correct GRS location. Thus, even if GRS writes are performed by a macroinstruction before it fails, the IP can be restored to its original state and a retry can be attempted.

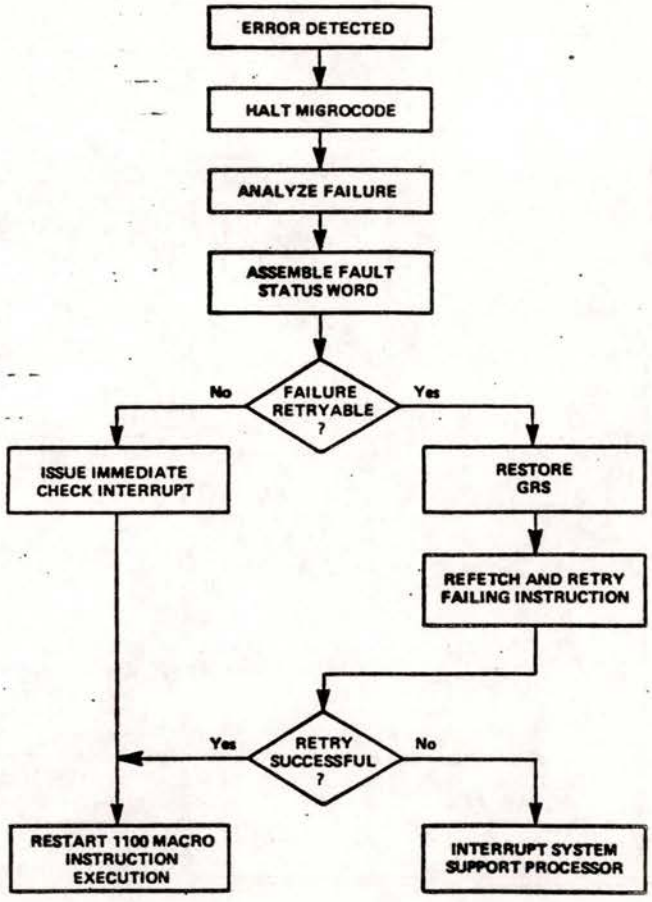


Figure 4. Flowchart of CPU Error Recovery

This approach permits retry of the vast majority of the macroinstructions executed in a normal program mix. If restoration is possible so that a retry can be performed, the failing macroinstruction is retried by restoring it into its original storage location and then refetching and reexecuting it. If the retry is successful, program execution continues from the point of the failure and a subsequent interrupt is presented to the operating system for logging purposes.

If the retry or the fault recovery microroutine fails, execution in the IP is halted and the system support processor is interrupted. Figure 5 shows the SSP fault recovery sequence. The IP is scanned to determine the machine state. If the failure is in the control storage, correction is attempted by rewriting the storage to its original values. If this is unsuccessful, control storage is rewritten using complemented values, and a designator is set which causes each control word to be reinverted before use. This allows both soft failures and single cell solid failures to be corrected. If the correction is successful, the IP is restarted and a second retry is attempted. If this retry is successful, program execution can continue.

If the retry is unsuccessful or if the original failure was not in the control store, the failure may still be recovered in a multiprocessor configuration by "program transplant." Utilizing this procedure, the SSP reads the program environment and GRS contents from the failing IP and transmits them to the operating system for use by another IP in the configuration. The failing instruction sequence can then be restarted in the other IP. Effectively, a failure in one IP can be retried in another IP.

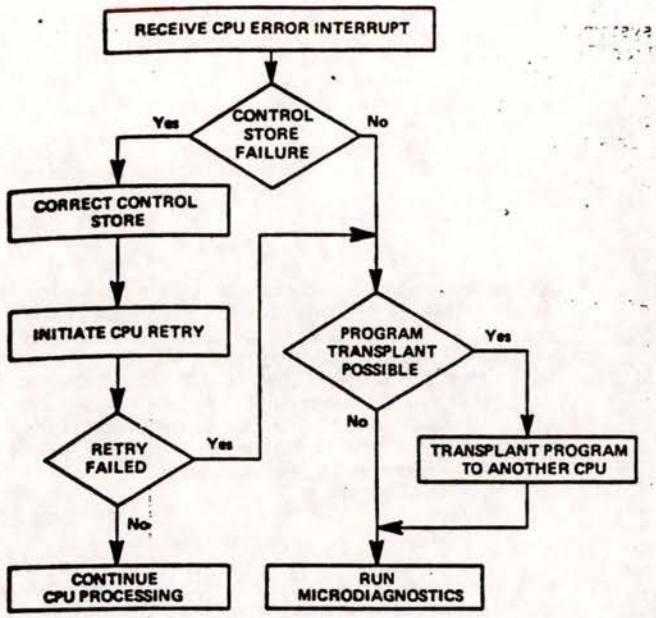


Figure 5. Flowchart of SSP Recovery Procedure

Following program transplant, or if no transplant is possible, the internal storages in the failing IP are reinitialized by the SSP, and a microdiagnostic program is invoked. If the execution completes successfully, the failing IP can be reintroduced to the system. If the microdiagnostics fail, the operator is notified and the maintenance techniques discussed in section G are used to isolate and repair the failure. No manual intervention is needed in the recovery procedure until recovery is as complete as possible and the SSP has determined that a solid, unrecoverable failure exists in the IP.

F. Fault Injection

In the 1100/60 processing system, a capability has been included that uses hardware and software to verify that the fault detection, isolation, and recovery mechanisms are operational. The capability is provided through fault injection, which is the process of deliberately causing a fault to occur in a system by inserting erroneous data or control signals in a portion of logic covered by a fault detection capability. The need for such a feature arises because the error handling portion of the design is not frequently exercised under normal operation of the system, so it could fail with no indication to the software. Without a periodic verification of the integrity of the error handling logic, one could not be confident of its ability to function when needed. This capability is also needed during prototype testing to verify the design of the hardware and software fault-handling capabilities.

The 1100/60 system incorporates fault injection for fault detection circuits in the processor, input/output unit, storage interface unit, and main storage unit. In the case of the control storage or the small storage used for instruction decoding in the IP, the fault injection is under control of the SSP. For example, the SSP can inject a control storage parity error by writing incorrect parity directly into the desired address location in order to stimulate one of the associated parity checkers. Injection of other IP faults is initiated by the presence of certain processor state bits and a predetermined micro-control store bit. During normal system operation, the state bits are set via a special 1100 macroinstruction which is periodically executed during system idle time. This instruction causes injection of a fault and then monitors for detection of that same fault.

After the fault is detected, the system is purged of the fault, and the instruction is retried without an injected fault. Successful completion of these steps indicates that the particular fault detection and retry hardware exercised is operating correctly.

In the input/output unit, injection is also under the control of the SSP. For example, a fault can be injected in the channel control word storage registers by setting and clearing flip-flops that specify the type of fault desired and the device operation during which the fault should occur. After the proper logic has been primed for the injection, the chosen fault will be triggered the next time the preselected device operation takes place.

In the 1100/60 cache (SIU) and main storage unit (MSU), the injection process is controlled by an IP instruction routine. Forced faults internal to the SIU are specified by the unit requesting or sending data, and that requestor then expects a certain type of fault at a predetermined point in the operation. In the MSU, the routine provides the capability to insert an invalid ECC code or bad parity on read data. On the access cycle of the MSU, the fault should be detected and an interrupt signaled.

G. Maintenance

The maintenance philosophy for the 1100/60 incorporates four methods of dealing with a failure in the system. In the order of priority of use, they are:

1. Automatic error log
2. Macrodiagnostic tests
3. Scan compare tests
4. Manual troubleshooting

The automatic error log represents a record kept by the system of any fault handled by the built-in fault detection/correction, isolation, and recovery mechanisms. In the majority of cases, the log should provide sufficient information for the customer engineer to determine the source of the problem.

In those cases when the problem has not been completely identified by the error log information, macrodiagnostic tests are employed. These tests are written on a macroinstruction level; they serve to exercise most portions of the system establishing either a high level of confidence that the system is operating correctly, or determining the general area in which it has failed.

If the previous two techniques have not identified the trouble area or if a finer resolution of the failure is needed, the scan-compare tests are run. These routines are tests run under the control of the SSP. They make use of the previously mentioned scan network (built into the processor), permitting access to the state of all major storage elements in the IP. The tests will exercise the processor and compare the results to a table of predetermined correct results to establish the nature and source of the problem. A similar process is applied to the IOU, which is a hardwired unit. I/O instructions are executed and the scan network is employed under control of the SSP without the use of the IP.

Accessible through the scan network is a built-in logic analyzer which is available in each 1100/60 as an additional diagnostic tool for the customer engineer. This feature permits automatic storage of 1024 consecutive states of any 16 logic points sampled twice during each microinstruction cycle. The logic analyzer is helpful in the diagnosis of a particularly difficult failure mode, such as might exist in the presence of an intermittent fault.

Should none of the three methods above locate the failure, the customer engineer will resort to manual troubleshooting techniques. These manual efforts, however, are greatly enhanced by the features available in the SSP. For example, the scan network will permit the troubleshooter to capture and display on the CRT a substantial amount of internal test-point information that in the past would only have been available by

using such tools as oscilloscopes and logic probes. In addition, the SSP has a communications capability that allows a linkage with a remote maintenance facility staffed with a team of diagnostic experts. This feature provides the ability to collect from a remote location any error information (for example, the contents of the logic analyzer) that is normally gathered and analyzed onsite. In addition, this interface can be used for remote control of diagnostic execution and troubleshooting procedures.

As a means of verifying the tolerance of the system to reasonable voltage variations during operation, the 1100/60 IP incorporates power supplies with output levels that can be varied by programmable margins under control of the SSP. This allows customer engineering personnel at the computer site, or at the remote maintenance facility, to alter the output voltages easily and to observe the IP behavior. Hence, this margin testing tool provides one means of identifying marginally stable components before they have degraded to the level where a system fault may occur.

IV. ARM EVALUATION

To measure how the design goals were being met with respect to the ARM characteristics of the IP, an evaluation method was developed that facilitates a quantitative prediction of the ARM behavior of the central complex. Using this method, an estimation was made of the fault detection/recovery coverage, the system stability or mean time between stops (MTBS), the mean down-time (MDT), the mean time to repair (MTTR), and the availability of the central complex.

The use of the evaluation method requires an initial examination of the various elements of the system and their anticipated contributions to the overall stability and availability of the central complex. After such an examination the two major ARM measures can be analyzed: MTBS and availability. MTBS is a measure of system stability and is calculated by evaluating its two components: hardware MTBS (MTBSH) and software MTBS (MTBSS).

The hardware stability is determined from component failure rates, the coverage of the fault detection mechanisms, and the recoverability of the system from each of the faults detected that do not lead to a system stop. The failure rates are obtained by studying vendor, government, and internal failure data for each of the integrated circuits and components used in the units. Coverage is analyzed by examining the current detailed hardware documentation for the system and determining which fault detection circuits will capture which faults, and in which chips. The recoverability factor is determined by studying what percentage of detected faults can be recovered from by each recovery mechanism and by analyzing the probability of success of that recovery.

The software stability is calculated by examining the inherent characteristics of the modules, such as size and complexity; considering the quality assurance during development; and evaluating the environment in which the software will be used based on our experience with the stability of similar software systems in the past.

The other major ARM factor, system availability, is predicted by considering the MTBFs and MTTRs of the various units in the central complex, the amount of redundancy of units in the system, and the recovery time necessary following a system stop. The application of the evaluation method described above proved to be a valuable tool for management and design personnel in gaining visibility of the unfolding ARM characteristics of the 1100/60 CPU versus the established goals for development. It has been encouraging to observe that preliminary data gathered from approximately one hundred initial installations reflect a very favorable comparison with the predicted values.

V. SUMMARY

As the applications of computers become more complex and sophisticated, and as new electronic technologies emerge in the industry, the demand, as well as the potential, for increased availability, reliability, and maintainability appears to be growing. The SPERRY UNIVAC 1100/60 Information Processing System reflects the results of a coherent development effort that takes advantage of the current state of the art in achieving a high level of inherent quality of design and a dramatic increase in the fault tolerant attributes of the commercial computer system.

BIBLIOGRAPHY

1. Sedmak, R. M., H. L. Liebergot, "Fault Tolerance of a General Purpose Computer Implemented by Very Large Scale Integration," Digest of Papers, FTCS-8, *The Eighth Annual Conference on Fault Tolerant Computing,* Toulouse, France, pp. 137-143, June 1978.
2. Borgerson, B. R., M. L. Hanson, and P. A. Hartley, "The Evolution of the Sperry Univac 1100 Series: A History, Analysis, and Projection," *CACM,* pp. 25-43, January 1978.
3. Borgerson, B. R., M. D. Godfrey, P. E. Hagerty, and T. R. Rykken, "The Architecture of Sperry Univac 1100 Series Systems," *Digest of Papers, Sixth Annual International Symposium on Computer Architecture,* Philadelphia, PA, pp. 137-146, April 1979.
4. "The Microarchitecture of Univac's 1100/60," *Datamation,* pp. 173-178, July 1979.
5. Stewart, J. H., "Future Testing of Large LSI Circuit Cards," *Digest of Papers of the 1977 Semiconductor Test Symposium,* Cherry Hill, NJ, pp. 6-15, October 1977.
6. Stewart, J. H., "Application of Scan Set for Error Detection and Diagnostics," *Digest of Papers of the 1978 Semiconductor Test Conference,* Cherry Hill, NJ, pp. 152-158, October 1978.

L. A. Boone; BSEE University of Wisconsin '68; Senior Logician, Logic Design Department, Major Systems Division.

H. L. Liebergot; MSEE Drexel University '71; B.S. Physics Drexel University '66; Technical Director, Oki Univac Kaisha, Ltd., Joint Ventures/Japan Department.

R. M. Sedmak; MBA Lehigh University '73; BSEE Lehigh University '71; Manager Availability, Reliability, and Maintainability - Development Program Management Department.

FAULT HANDLING ON THE IBM 4341 PROCESSOR

Mark L. Ciacelli

IBM Corporation
System Products Division
Endicott, New York 13760

Abstract

This paper discusses the error-detection, fault-isolation, and recovery mechanisms used in the IBM 4341 Processors. It includes a description of the hardware-checking circuitry and the microcode/hardware designed to respond to errors detected by this circuitry. A Service Processor subsystem in the 4341 is used for error recording, error analysis, and fault isolation. The Service Processor is also used to re-initialize the processing unit so that it can attempt (via dedicated hardware) to retry the machine function terminated by the error condition.

Information about unrecovered errors is reported to the system control program which may initiate recovery procedures to minimize the effects of these errors.

An added aspect of the retry design is that some of the same hardware/microcode is used to eliminate much of the performance overhead of the System/370 virtual addressing function. A description of how this can be accomplished is included.

Also included is a brief discussion of a microcoded test tool that was used to functionally test the retry mechanism.

Introduction

There are two basic parts in the 4341 system structure, the PU (Processing Unit), which includes the channels, the CPU (Central Processing Unit), and storage, and a microprocessor-controlled SP (Service Processor). Both of these subsystems are microcode-controlled. The SP handles many functions including initial power bring-up, Operator Console Functions, and Remote Support Facilities (RSF), as well as Error Handling and PU initialization. Much of this is made possible through the use of a design system called level sensitive scan design (LSSD). The LSSD technique was originally implemented to solve the problem of inaccessibility of internal signals in large scale integration (LSI) technology [1].

The only type of storage element (other than arrays) allowed in LSSD is the shift register latch or SRL (throughout the rest of the paper these SRLs may be referred to simply as 'latches'). All the latches on one LSI chip are connected together to form one long shift register. Chips related to a specific area of the machine are then connected together to form an even longer shift register sometimes referred to as a 'scan string.' In the 4341 these scan strings are organized into four general areas: clocks, PU and channels, storage, and control storage. Even though LSI prevents the probing of individual latches on a chip,

LSSD enables a test system to control and retrieve the contents of any storage element on the chip by means of a simple shift technique, often referred to as 'scanning.' This idea was originally used to test for dc faults using computer-generated test data. However, as the 'Service Processor' section of this paper will show, the same philosophy can be implemented in a slightly different manner and applied to error handling and recovery.

What follows below is a scenario of how the machine detects a malfunction and what happens throughout the system (including software) after the malfunction is detected. These sections will cover error detection and recovery. Fault isolation, page fault handling, and the retry test tool are discussed in following sections.

Error Detection

The effectiveness of any error handling and recovery scheme is very dependent upon how well the hardware can detect machine errors. On the 4341 there are basically three ways in which the hardware checks for malfunctions. The first, and simplest way to check for errors is to duplex the line, that is, send two copies of the same line and check to see both lines are the same at the receiving end. This is done with a single exclusive-OR gate. The two signal lines are sent out of phase so that an error condition is recognized if both lines are open. The second method is to include a parity bit with the data bits and to check the parity with a string of exclusive-OR gates. The 4341 is an odd-parity machine, meaning that if an even number of bits (data and parity) are on, an error latch will be turned on to indicate a malfunction. The third method of error detection is a check for special error conditions. This can be a check for an invalid code in a command register or an invalid combination of control lines. In general, control lines are checked by duplexing or by testing for invalid combinations of multiple control lines. All critical control lines are duplexed to prevent error propagation. It is also a general rule of thumb to duplex all control lines that come off the card. Data lines can use any of the detection mechanisms described above.

When a malfunction is detected by using one of these mechanisms, an appropriate error latch is turned on. The error latches remain on until they are scanned off by the SP. This 'freezing' technique is extended in the CPU and channel areas of the hardware. In these areas, not only is the error latch frozen, but the register that contains the bad parity is also frozen. This was done to aid the initial hardware debug as well as to make error isolation easier by not allowing the invalid data to be overlaid. Again, the only way to reset these registers is to scan in valid data. In the case of an error latch being set incorrectly, there

is a provision for disabling that particular error latch by setting another 'scan only' latch that has the specific function of disabling a given error latch. In this way an incorrect error indicator will be ignored until the error checking hardware itself can be fixed.

The Service Processor

The SP is constantly polling the status of the PU at regular intervals to detect the occurrence of a malfunction by scanning a summary machine check latch. At the time of the malfunction, the SP scans out the contents of the latches using the shifting technique previously described. This data is analyzed by the SP to determine the appropriate recovery action to be taken when the PU is restarted. There are several factors considered when making this determination. In System/370, two types of processing can occur simultaneously - CPU (Central Processing Unit) and channel. When a CPU operation is affected by a malfunction, either the operation is successfully reexecuted by error retry or the operation is terminated and a damage report is given by the machine check interrupt process. When a channel operation is affected by a malfunction, the operation is terminated and a damage report is given by the I/O interrupt process. Channel recovery is not attempted by the machine since most channel operations can be reexecuted by the system control program.

The 4341 checkpoints so that the machine instruction that was in progress at the time of the malfunction can be reexecuted. Instructions are retried by restoring information that is continuously saved and periodically purged by hardware. Saving and purging information for retry does not affect the performance of the machine. In general, retry cannot be attempted if a facility (latch or array) that is available to the microcode for storing information between instructions is altered by an instruction after the facility has been sourced, or by the effect of a malfunction unless each such facility has been saved by the retry hardware. Interrupts are retried without saving and restoring information. As a result, interrupt retry cannot be attempted as frequently as instruction retry.

It follows from above that each machine check latch must be categorized as retriable or unretriable depending on the extent of the damage associated with malfunctions that caused that latch to be set. This categorization is done during the design stages by both hardware and microcode designers.

The appropriate recovery action is initiated by scanning a modified system reset state into the PU. The system reset state is modified to preserve selected latches and to pass control information to the PU. The restoration of arrays in the machine is handled by the PU for performance reasons.

After the modified system reset state is scanned in, the SP starts the PU clocks running. The SP then saves the machine check data on the system diskette as part of the error log associated with this particular malfunction. The logging of error information is done after the error analysis and re-initialization in order to speed up error recovery and get the PU running again.

The Processing Unit

Reexecution of the failing instruction requires a second, or 'retry', array for each functional array. The retry array saves the address and data of each update to the functional array. In the 4341 there are two such functional arrays, the local store array, which contains the GPRs (general purpose registers) and FPRs (floating point registers), and the Cache Array, which is used as a fast-access storage area. If, as a result of it's analysis, the SP indicates to the PU that any particular array(s) should be restored, the PU will use the appropriate retry array(s) to restore the functional array(s) to the correct state. The restoration of arrays could cause problems if the arrays contain bad parity, since these arrays must be sourced before any destinations are made. The restoration process, as a result of these sources, could cause a second machine check. To avoid this problem, special consideration was given to ignoring bad parity during the restoration process [2].

At this point, the machine attempts to retry the failing instruction. If it is successful, the PU informs the software with a Machine Check Interrupt Code (MCIC) and continues normal operation. If, for some reason, the retry attempt is unsuccessful, the MCIC also contains information to aid the software in it's own error handling procedures. An attempted retry could be unsuccessful if the malfunction is a stuck fault, if there is invalid data that cannot be restored to a valid state because the original data is no longer available, or if there is a second, unrelated, machine check encountered during retry.

The PU uses the data passed from the SP to build the MCIC and it stores the field in a fixed storage location so that the software has access to it. Different parts of the MCIC are assigned various functions [3]. The Subclass (bits 0-5,7,8) identifies the machine check conditions causing the interruption. At least one of these bits will be on, unless multiple errors occur, in which case several bits may be set on. These bits include System Damage, Instruction Processing Damage, System Recovery, Timer Damage, Timing Facilities Damage, and External Damage, as well as a few others (the use of these is discussed below). Storage Errors (indicated by bits 16-18) effect either some part of main storage or a storage protection key. The Failing Storage Address, if

marked valid, identifies the address of the storage location in error or the storage block associated with the storage protection key in error. The Validity Indicators (bits 20-31,46,47) are each associated with a particular facility in the machine. These include the PSW, Failing Storage Address, GPRs, FPRs, Control Registers, CPU Timer, and Clock Comparator. When a validity bit is one, it indicates the field was logically valid at the time of interruption and no error was detected when the data was stored. If the bit is zero, the information is unpredictable and steps should be taken by the software to ensure valid data is placed in the facility.

There are two types of machine checks that do not generate a MCIC. The first of these is any machine check that occurs while the retry microcode is executing. In this case, a MCIC is generated for the original error, but not for the second malfunction. The second case is when a malfunction has been isolated to an I/O instruction and the Condition Code is set to one. In this second case the CPU has not suffered any damage, therefore there is no need to create a MCIC. The Condition Code setting of '1' says that a CSW (channel status word) has been stored and when the operating system checks the CSW it will find that a Channel Control Check has been posted and will retry the channel operation that was in progress when the malfunction was detected.

Operating System Recovery

If processor retry was successful, the MCIC will indicate System Recovery and the operating system will continue normal operation. If the machine fails to retry an instruction successfully, the operating system's machine check handler is notified and attempts to keep the system running as fully operational as is possible after analyzing the MCIC data. Different operating systems will handle machine checks differently, but for the purpose of discussion we will consider the IBM VM/370 operating system.

Recovery can be classified as one of the four following categories [4]. Function Recovery is recovery without damage to the system or the interrupted user. This is done by correcting storage protection keys and intermittent errors in real storage. If Functional Recovery is impossible, the next step is System Recovery, which is the continuation of system operations, at the expense of the interrupted user, whose virtual machine is terminated. This is contingent upon the fact that the user is not critical to system operation. Third, if an error causes the loss of system integrity, an Operator-Initiated Restart is required. The fourth and least desirable of the recovery options is System Repair which requires the services of maintenance personnel.

Initially, the VM/370 Machine Check Handler (MCH) checks the MCIC data for what it considers uncorrectable errors. These include System Damage, Timing Facilities Damage, MCIC equal zero, and PSW invalid. Any of these conditions will result in the system entering the wait state after the error is recorded and the primary operator is notified.

If Instruction Processing Damage is indicated it is possible that retry has failed or was not attempted at all. In this case, if the processor was in Supervisor State, the error is considered uncorrectable and the VM/370 System is terminated. If the processor was in Problem State, the virtual machine is reset or terminated and the system continues normal operation. Instruction Processing Damage could also mean either a multiple-bit error in main storage or a storage protection key is in error. In either of these cases the VM/370 MCH must determine whether the error is intermittent or solid. If the error is solid, the MCH removes the affected storage area as an available system resource or terminates the VM/370 System if the affected area is associated with the VM System Control Program. If the affected area is not being used by the Control Program, the virtual machine that was using this storage area is reset or terminated. If the error is determined intermittent for the storage protection key error, the key is reset and operation continues normally for the entire VM/370 system. If a storage error is determined intermittent, the associated page frame is marked invalid and the virtual machine is either assigned a new page frame or is reset (or terminated), depending on whether or not the damaged storage area was altered by the virtual machine. In either case of a storage error, system operation continues normally for all other users.

Fault Isolation

Concurrent with the system recovery from a machine malfunction, the SP performs another of it's error handling functions, that being fault isolation. Fault isolation is accomplished by means of a series of SP-microprograms, Error Log Analysis (ELA), which analyzes the error log that has been recorded on the system diskette. FLA will generate a code to indicate the defective FRU (Field Replaceable Unit) for either intermittent or solid failures. This strategy provides a considerable time savings in problem resolution. It reduces the need to recreate the failure with a diagnostic, provides parts information to the C.E. (customer engineer) prior to arriving on-site, eliminates time consuming methods of capturing the failure, and reduces the number of parts and personnel required to correct the failure.

The algorithm that is used by ELA to pinpoint the area of the machine that is in error has a 'pyramid' type of arrangement. That is, it first tests the areas that could

possibly propagate an error into different areas of the machine. This eliminates the possibility, for example, of finding an error in the PU, replacing the appropriate FRU(s), then finding out later that the PU error was actually a result of a bad Clock chip whose error had been propagated down through the internals of the CPU.

Page Faults

One of the added advantages of error retry design on the 4341 is that much of the performance overhead of the System/370 virtual addressing function can be eliminated by using some of the retry hardware/microcode. This type of function is referred to as 'access retry' as opposed to 'error retry' which has been discussed above. The way access retry works is that when a page fault is encountered, the page fault interrupt handler calls the subroutine of the PU-Retry microcode which restores the local store and cache arrays to make it appear as though the instruction had never occurred. A program interruption is then taken so that the software can eliminate the page fault by fetching the page from the paging device. The paging software then causes the interrupted instruction to be reexecuted. This feature eliminates the need for pre-fetching all of the data for all storage accesses. This turns out to be a considerable savings, even though it increases the execution time of those instructions that incur page faults. This savings is possible since only a very small percentage of storage accesses require a page fault.

Retry Test Tool

The final section of this paper is devoted to a brief discussion of the microcoded test tool that was used to functionally test the retry mechanism. On many previous systems, retry was tested by using an electronic test box to physically inject errors via hardware probes [5]. This test box was used in conjunction with a software test program that consisted of a test case monitor and a series of test case programs. Each test case was looped through as many times as there were machine cycles in the test case. One error was injected for each machine cycle, that is, on the first pass, the error was injected on the first cycle, on the second pass, the error was injected on the second cycle, and so on. The test case monitor controlled the error injection unit during the operation.

Once again, we have taken advantage of the LSSD scan strings and provided the same function without the hardware box by using the SP to access the scan strings. The first run through the test case is error-free to allow the SP to compile a list of the control storage addresses of all the machine cycles executed in the test case. This list of addresses is passed to the test case monitor. A special

machine instruction is issued before each error pass through the test case to provide the control storage address of the machine cycle on which the error is to be injected. The selection of latches to be manipulated in order to inject the error (by simulating a hardware check) is done by SP-microcoded routines, one of which must be selected by the operator at test execution time. The SP stops the PU on the specified control storage address, calls the selected microprogram to inject an error, and restarts the PU. This sequence is repeated until errors have been injected for each machine cycle of the particular test case.

This testing technique has several advantages over previously implemented schemes. There is no need to build an external box to match each machine's technology and there is no longer any risk of disrupting switching characteristics of the hardware by physically attaching probes. As many tapes and/or diskettes as are needed can be generated and run on any number of machines, unlike past systems which required one box for each machine being tested simultaneously.

Summary

Fault-tolerant computing has come a long way in a short time, from simple voting circuits that will detach faulty hardware modules to sophisticated hardware/microcode/software recovery mechanisms described here. The reliability and availability of the computing system is greatly improved with this scheme of error detection, correction, and analysis.

Acknowledgment

The concentrated efforts of many people, including hardware designers, microcoders, and testers has been essential to the development of the concepts described in this paper and I would like to extend my thanks to all who helped in the collection of information.

REFERENCES

[1] N. C. Berglund, "Processor Development in the LSI Environment," Electronics, March 15, 1979, McGraw-Hill, Inc. 1979.

[2] S. P. Dale, G. L. Hicks, K. F. Pezdirtz, "Restoration of Arrays Containing Bad Parity," IBM Technical Disclosure Bulletin EN879-0039, Vol. 22 No. 5, October 1979.

[3] IBM System/370 Principles of Operation, GA22-7000-5.

[4] IBM Virtual Machine Facility/370: System Logic and

Problem Determination Guide, SY20-0886-1.

[5] S. P. Dale, G. L. Hicks, K. F. Pezdirtz, "Micro-coded Retry Test Tool," IBM Technical Disclosure Bulletin EN877-0216, Vol. 21 No. 6, November, 1978.

IBM 3081



Figure 7 Cross section of the thermal conduction module.

Table 2 Percentage distribution of total wire count by packaging level.

Packaging level	*3033 Processor*	*3081 Processor*
Chip	63.0	94.1
Module	—	5.4
Card	33.2	—
Board	2.2	0.4
Cable	1.6	0.1
	100%	100%

Table 3 Percentage distribution of total wire length by packaging level.

Packaging level	*3033 Processor*	*3081 Processor*
Chip	1.3	9.2
Module	—	41.9
Card	38.8	—
Board	12.9	31.1
Cable	47.0	17.8
	100%	100%

compared to that previously existing at the card and board levels of packaging. Table 2 shows the percentage of total wires counted, at each level of packaging, in the IBM 3081 and 3033 processors for an equivalent amount of logic. Table 3 shows the percentage of total wire length at each packaging level.

• *Module cooling*

A 3081 module generates a power density (heat flux) of the order of 10^5 W/m^2. This is about a tenfold increase compared to the heat flux generated at the module level in the 3033 processor, which uses forced-air cooling over the modules and interboard heat exchangers to cool the air. Extensions to this cooling technology (such as increasing forced-air flow) were not suitable for removing the heat flux of the 3081 module; a new cooling method had to be used.

Basic thermal problems associated with a high-performance package are the transfer of heat from the source (*e.g.*, the chip) to an external package surface, the removal of heat from the external surface by a coolant, and the maintenance of coolant temperature. The above problems were addressed in the development of the *thermal conduction module* (TCM).

A cross section of the thermal conduction module is illustrated in Fig. 7. The TCM has a helium-filled module cap which provides a thermal conduction path from the back of each chip, via one contacting piston per chip, to the cap. The cap is in contact with a water-cooled cold plate (water jacket) which is removable for module service without disconnecting the supply of circulating water. A separate coolant distribution unit is used to pump water (at approximately 24°C) through the cold plate. The TCM maintains a circuit-junction temperature in the range of 40°C to 85°C and provides a cooling capability which exceeds the requirements for the 3081 processor.

The paper by Chu, Hwang, and Simons [7] in this issue describes the thermal problems, the design alternatives considered, and the theory of operation of the TCM using one-dimensional models. The paper by Oktay and Kammerer [8] in this issue discusses the thermal, mechanical, and environmental experiments, as well as the extended three-dimensional analytic modeling required to develop the TCM.

Reliability, availability, and serviceability (RAS)

The lower cost per circuit of LSI technology as well as its intrinsic reliability provides potential for improved system RAS. The design of the 3081 processor complex includes added function for error-detection and error-correction mechanisms, as well as for hardware error recovery and reconfiguration. Additionally, serviceability is improved through a new approach to diagnostics.

The intrinsic reliability of the 3081 LSI package is a major factor in the reduction of solid failures compared to those experienced using previous technologies. The intermittent type of failure, which has always been a problem, now becomes more important and requires an approach to error detection and fault isolation considerably different from the error re-creation strategy used in predecessor large-scale IBM systems.

In past systems, an error re-creation strategy was satisfactory because conventional tools and human intelligence were sufficient to isolate and repair the fault. But, within the LSI package used in this system, access to chip I/O pins is

prohibited—in fact, the field engineer does not have access to the TCM I/O pins.

Thus, LSI technology requires an approach to fault isolation and service different from that used in past systems; but LSI technology also offers the opportunity to develop a different approach with new considerations for logic design, packaging, and processor controller functions.

• *Error detection*

The design of the 3081 processor places greater emphasis on instantaneous error detection and associated analysis procedures than has been the case in previous IBM large processor systems. The strategy demands that error checkers be designed and located so that the logical state of the processor (in terms of "1" and "0" levels) is captured at the time of the error.

It takes several cycles to stop the clocks of the hardware element which has reported the error. The use of arrays with high density and speed makes it possible to capture all state data necessary to isolate a fault which has been detected in the processor. Shift latches using Level-Sensitive Scan Design (LSSD) [9] permit data to be scanned out to the processor controller for analysis. The Processor Unit contains 126 scan rings, containing a total of almost 35 000 bits of data, which are read out in less than 100 ms. In addition to permitting data to be scanned out, LSSD allows data to be scanned in to set the state of the processor to any predetermined value.

• *Hardware error recovery and reconfiguration*

After fault-related data are scanned out to the processor controller, an attempt is made to recover from a hardware error by resumption of processing of the interrupted instruction execution sequence. The instruction element is designed to establish a checkpoint at approximately every 10 to 20 instructions, to which the state of the machine may be restored in order to retry instructions. Extensive use is made of the 32 × 18-bit 7-ns array chip to save the contents of altered general registers, floating-point registers, the program-status word (PSW), and buffer storage. Up to eight attempts are made to retry instructions.

In the event that there is a solid failure, the 3081 Processor Unit can be reconfigured. The dyadic organization makes it possible to logically remove a failing central processor and to continue processing with the remaining central processor until the customer chooses to schedule maintenance. Similarly, a portion of central storage, a single channel group, or a single channel interface may be logically removed from the processor organization while still permitting processing to continue, although system performance may be degraded.

• *Diagnostics and serviceability*

The diagnostics of the 3081 differ considerably from those used with previous large IBM processors. A key objective was to isolate to as few TCMs as possible per failure, ideally only the failing one. Another objective was to isolate the failing TCM (which is the field-replaceable unit), on the basis of the data captured at the time of error detection, for a very high percentage of failures. Analysis routines, which run automatically and concurrently with customer operation, identify a single failing TCM a high percentage of the time. Full isolation to a FRU is achieved for solid failures by using validation tests, automatically called when required by the processor controller, which detect and isolate the type of fault in which logic is stuck at a "1" or at a "0" level. Test data are entered into the processor via the LSSD scan-in data paths. The validation tests are run by the field engineer.

As stated previously, the RAS strategy for the 3081 Processor Unit must be, and is, significantly different than for previous large systems. The approach used for *error detection* and *fault isolation* (ED/FI) and the model used for projecting the ability of the system to detect and isolate failures are described in the paper by Bossen and Hsiao [10]. The paper by Tendolkar and Swann [11] presents the concepts of automated diagnostics implemented for the 3081 and describes the technique used to estimate the isolation effectiveness of the diagnostic package. Integral to this RAS strategy is the implementation and function of the processor controller, which was previously mentioned and which is discussed in detail in Ref. [2].

Computer support

The dependency upon computer support for physical design and test generation is well known [12]. The application of LSI technology has further extended computer support through new requirements for physical design [13], the automatic generation of tests, system design aids, and the control and tracking of complex manufacturing processes.

• *Engineering design system*

Most of the computer support is provided by an *engineering design system* (EDS), which includes programs that check for wireability and that generate tests for each packaging level, and which also generate the physical design of shapes, patterns, precise locations, and interconnections to complete the total fabrication process. These programs, and others used for design and analysis, share a substantial amount of engineering design data. The use of one system, EDS, to maintain the data in a set of interrelated data bases reduces human intervention in data transcription, increases data integrity, and avoids duplication of resources. Extensive use was made of computer support and EDS data throughout the 3081 development effort and continues to be a significant factor in the manufacturing process.

contains 72 array chips. A vertical partition (one 32-bit word on each TCM) was adopted to achieve the machine-cycle-time objective.

The store-in-cache algorithm was selected because it produced a higher internal performance level than a store-through algorithm, when the characteristics of the system were considered in conjunction with the central storage access time. Modeling results indicated that a store-in-cache algorithm yields superior performance as soon as the central storage access time exceeds the range of ten machine cycles. The 128-byte line size was selected since it delivered the optimal performance when analyzed with the central storage, system controller, and dual-processor characteristics. Store-in-cache also provided a foundation for a checkpoint-retry algorithm for the CP, which will be described subsequently.

The *control store element* (CSE) controls the sequencing of the horizontal microcode of the CP. The CP microword is 108 bits wide and has an addressing capability of 32K microwords. Two microword formats are supported, one for each of the IE and the VFE. A 1K-microword writable static array contains the most-frequently-used microwords. A second 1K-microword array contains 32 lines of 32 microwords each. It is fully associative with a least-recently-used replacement algorithm, and lines are dynamically loaded on demand from a "hardware system area" located in a portion of central storage. Like the cache, the control store arrays are packaged on two unique TCMs with a vertical partitioning; however, a significant number of controls had to be replicated to achieve the cycle-time objectives.

A unique feature of the CP is its hardware checkpoint-retry mechanism. The basic concept is to establish a checkpoint at some instruction *N*. The contents of the architected registers (PSW, GRs, etc.) are saved in backup registers. As instruction execution proceeds, any store into the cache will cause the old value of a changed doubleword to be saved in a push-down array. In the event of an error condition, the architected registers will be restored to their original value. The contents of the cache will also be returned to its original values by storing back the contents of the push-down array in the reverse order. Regular checkpoints are established during normal instruction execution under specific machine conditions with a negligible effect on CP performance.

• *System controller (SC)*

The major function of the SC is to provide the paths and controls for the communications between the major functional units and with *central storage*. A unique structure was adopted due to the packaging limitation (I/O pins), the store-in-cache algorithm of the CPs, and the support of the dual-processor configuration with minimum performance degradation. The I/O needs of central storage, the SC, and the EXDC provided the requirement for the six-TCM board with its additional I/O capability.

The implementation of the array portion of central storage is on a card-on-board package to achieve the density and cost requirements. Each board contains 4M bytes of central storage, which constitutes a *basic storage module* (BSM). The BSM is configured so that a 128-byte line can be accessed (read or write operation) with a single array operation.

The conventional doubleword interleaving of current large processors is effectively accomplished within the 4M-byte BSM. Of significant importance is that the bandwidth requirement for independent doubleword operations with central storage has been significantly reduced. This is the result of the store-in-cache algorithm of the CPs in conjunction with a 128-byte line, and the 256-byte (per channel) buffering capability for data within the EXDC. Thus, the SC was structured for a line-transfer characteristic. The basic data-bus width of all units connecting to the SC is eight bytes (bidirectional) with a data transfer rate of eight bytes per machine cycle.

To achieve a balanced utilization, a 2K-byte address interleave structure across the 4M-byte BSMs was adopted. This increment was chosen to minimize the complexity of memory reconfiguration and was sufficient to achieve a balanced load.

The store-in-cache algorithm has created a data integrity problem which is managed by the SC. Due to changed data residing in the caches of the CPs, the content of central storage is not always valid. The SC maintains a duplicate copy of each CP cache directory and determines if the central storage request will return valid data. In the event of a conflict, the SC initiates the casting out of the specific line of data from the cache to central storage to then be refetched by the requesting unit.

Each line of data, as it resides in a CP cache, is tagged with a *read-only* (RO) or *exclusive* (EX) status. A line must have EX status before a store operation can be complete within that line. To control simultaneous updates to a specific line by the CP or channels, the SC permits only one copy of a line to exist with EX status. Conversely, multiple copies of a line having RO status can exist simultaneously for use as long as no store activity occurs. If a store must be performed into a line with RO status, a change is effected by the SC to EX status, with the purging of the copy as necessary.

An algorithm predicting the usage (fetch or store) of a line of data was adopted to minimize the performance

John Reilly
Arthur Sutton
Robert Nasser
Robert Griscom

Processor Controller for the IBM 3081

The IBM 3081 represents an important step in the evolution of large-scale data processing systems. The incorporation of LSI technology in its design has resulted in a departure from the use of traditional support tools and techniques. Its increased processing capabilities and its dyadic organization have further accented the requirements for high availability and ease of operation. This paper describes the Processor Controller unit of the 3081 Processor Complex, which handles the functions of maintenance, monitoring, and control of the system. It discusses how this unit has dealt with these changes to keep pace with the demands of advanced technology and improved system availability. In providing the necessary test and support functions of reset and manual control, the Processor Controller exploits the level-sensitive scan design capability of the 3081 to obtain read/write access to all latches and arrays. This design approach, coupled with significantly expanded capabilities for error recovery, configuration control, and diagnostics, has significantly affected the structure and capabilities of the Processor Controller.

Introduction

This paper discusses the structure and functional capabilities of the Processor Controller for IBM's 3081 Processor Complex [1], which contains the first IBM large-scale processor unit [2] implemented in LSI. The technology used and the internal structure of the 3081 Processor Unit represent significant new characteristics that influenced the design of the Processor Controller. The 3081 Processor Unit, comprised of two central processors, 16 or 24 channels, and up to 32M bytes of main storage, is supported by a single 3082 Processor Controller which is dedicated to the performance of functions associated with the monitoring, controlling, and maintaining of the system. In providing these functions, the Processor Controller exploits the Level-Sensitive Scan Design (LSSD) [3] capability of the machine. The LSSD capability, together with a dedicated path to main storage and thorough visibility to power/thermal components, provides the necessary Processor Controller interfaces into the system.

In this paper we review the design objectives and the development process associated with the 3081 and discuss the impact that these had on the design of the Processor Controller. The LSI hardware test cycle, which had a significant influence on the Processor Controller organization, is examined, the 3081 system-level objectives that had to be met are discussed, and the structure and functions of the Processor Controller hardware and microcode are described.

Processor controller development

A basic objective of the 3081 development effort was to bring up the hardware in several stages in order to gain early experience with the new *thermal conduction module* (TCM) technology. To achieve this, it was decided that three discrete successive steps would be taken:

1. *Module test*—which placed a single TCM in a controlled environment with access to all of its input/output pins. Tests were applied, with the logic cycled and monitored for correct results. Probing of the individual chip input/output pads was permitted in this environment.
2. *Subsystem test*—which packaged a number of TCMs on their respective boards to form the various components of the Processor Unit (central processors, channels, and

Copyright 1982 by International Business Machines Corporation. Copying is permitted without payment of royalty provided that (1) each reproduction is done without alteration and (2) the *Journal* reference and IBM copyright notice are included on the first page. The title and abstract may be used without further permission in computer-based and other information-service systems. Permission to *republish* other excerpts should be obtained from the Editor.

system controller). Tests were applied in this environment, with all of the signal lines to and from the components controlled and monitored.

3. *System test*—the final stage of hardware bring-up, where all of the components were successively combined to form the complete 3081 Processor Unit.

In all of these steps, the Processor Controller was used as the primary interface into the hardware. This permitted a staged development of this unit and an ongoing analysis of its effectiveness. This also required that the Processor Controller hardware be of a proven technology; thus, existing hardware components were utilized whenever possible to reduce the Processor Controller development and test time.

In addition to these requirements, a set of fundamental objectives for the Processor Controller were established and stressed to offset some of the potential negative factors associated with the TCM package—factors such as high parts cost, no scoping ability, tighter power tolerance, and less physical partitioning. These factors resulted in the following objectives.

1. *Complete visibility* to the logic of the TCM was essential to the engineering bring-up of the hardware, since scoping of the logic was virtually impossible. This was also required to meet the very rigorous objective for diagnostics in the field. To hold repair costs down it was essential to be able to diagnose problems to the minimum number of TCMs.
2. *Logical isolation* of the components and/or elements of the 3081 was required to permit continued operation in the presence of solid problems. Concurrent analysis of the problem was also required to minimize the impact on availability. This logical isolation was very important since the density of the hardware package prevented the desired amount of physical partitioning.
3. *A single point of control* was desired to remove operator complexities associated with the partitioning of the Processor Unit. The operating system was established as this point of control, and this necessitated a communication path between the operating system and the Processor Controller.
4. *Power/thermal monitoring* was required, to the extent that disturbances in these areas could be detected directly, to avoid changing expensive TCMs when in fact a problem might be caused by a power fluctuation or "glitch." Early in the development cycle it was also projected that the TCMs would require very tight tolerance on power. However, as development progressed, the requirement for tight tolerances was relaxed.
5. *Expanded Processor Controller functions* were necessary to improve on system availability and maintainability to allow the 3081 to be maintained by personnel with knowledge of its physical packaging but not necessarily of its internal workings. With more intelligence built into the Processor Controller, the efficiency and cost of maintaining the 3081 would be improved.
6. *Extensive use of microcode* in all areas of the 3081 was desirable to avoid the long turn-around time associated with hardware changes. This objective, coupled with the relatively low density of the arrays in the TCMs, prompted the requirement for a dynamically loadable microcode structure [2]. This structure is discussed in more detail subsequently in this paper. The extensive use of microcode was of paramount importance to the Processor Controller development because the flexibility afforded by microcode was essential in adapting to the various test phases as well as for the quick resolution of problems encountered in the testing of the LSI elements.

Hardware structure

The physical organization of the 3081 Processor Complex is shown in Fig. 1. It is composed of three major units: the *electronic frame* (Frame S), the *power frame* (Frame X), and the *Processor Controller frame* (Frame E/J). The electronic frame contains LSI hardware for the two *central processors* (PROC boards CP0 AND CP2), the channels (EXDC board), and the *system controller* (SCUB board)—which is the path for CPU, channels, and Processor Controller to main storage. *Main storage*, not shown, is controlled by logic resident on the SCUB and EXDC boards. Power for this electronic frame is provided from the X frame.

The E/J frame contains the Processor Controller, which consists of a *functional controller* and an array of adapters required to provide the interface and storage media essential to the Processor Controller's role. The functional controller contains 128K bytes of storage and an eight-level hardware interrupt structure. For communicating to operating personnel, both on-site and remote, the *display cluster adapter* (DCA) and the *communication common adapter* (CCA) have been provided. The *Processor Controller file adapter*, a large-capacity fixed file, and the *auxiliary storage adapter*, a 512K-byte monolithic storage device, are used to hold the Processor Controller microcode and system diagnostics. The auxiliary storage adapter is also used as a paging device for performance reasons. The *diskette adapter* is used for reading engineering-change data from portable diskettes. All of this hardware was previously designed and built for other IBM products.

Simple hardware paths to the system were established, as shown in Fig. 1, by developing a *monitoring and system support adapter* (MSSA) through which a number of *logic support stations* (LSSs) and *device support stations* (DSSs) could be attached to the controller. In this configuration, the MSSA acts as a multiplexor, gating interrupts and

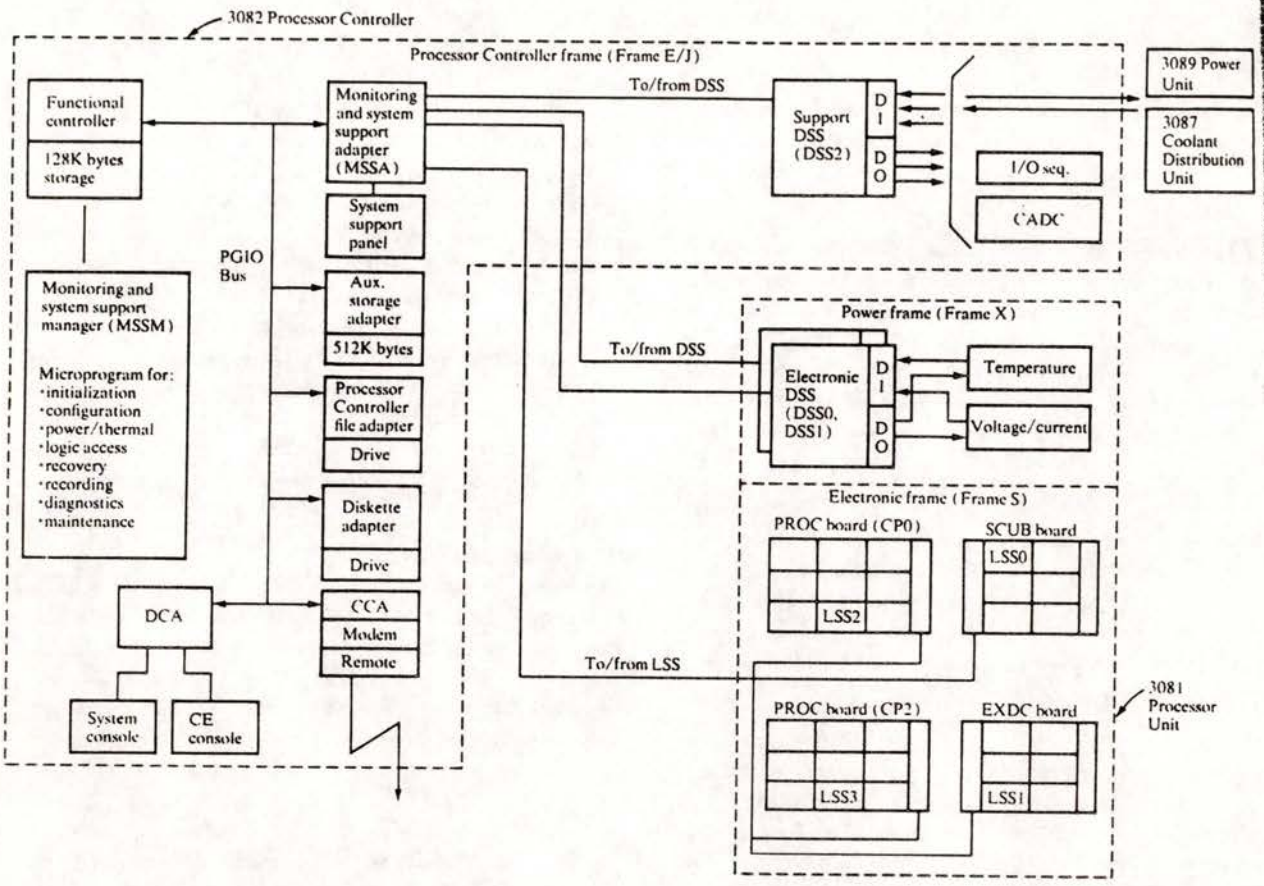


Figure 1 Physical organization of major units of the IBM 3081 Processor Complex. (Note: DCA – Display Cluster Adapter, CCA – Communication Common Adapter, LLS – Logic Support Station, DSS – Device Support Station, PGIO Bus – Programmed Input/Output Bus, DI – Data In, DO – Data Out, CADC – Central Analog/Digital Converter.)

commands between the Processor Controller and an appropriate support station. The LSSs are packaged in the TCM technology, have common logic functions, and provide a path for the Processor Controller into the hardware, with the following basic functional characteristics:

1. *Scanning* capability for the logic of the TCMs, using the LSSD scan-in scan-out method. This technique provides thorough read/write capability for all TCM circuits and arrays.
2. *Partition and clock control* for the components and/or elements of the system. Manipulation of a control register in the LSS, by means of Processor Controller microcode, permits isolation and separate clock control of the various parts of the system.
3. A *read/write of main storage* capability is provided for the Processor Controller through the LSS associated with the system controller (LSS0). This permits the Processor Controller to dynamically access main store without impacting processor and channel activity.
4. *Interrupts*, and associated interrupt codes, from the functional logic and error checker of the TCM hardware are presented to the processor controller through an intermittent path capability.

The DSSs are packaged in card-on-board technology, are designed with common logic functions, and provide control and visibility of the power and thermal elements of the system. Because of the lower reliability of the power and thermal sensors relative to that of other parts of the system, redundant paths were provided to permit the verification of an event through alternate means. The capabilities provided by the DSS hardware are the following:

1. *On/off control and sensor readout* are provided for all components of the power/thermal system.

2. ***Digital adjust*** capability for regulators is provided through the DSSs. That is, the output of each regulator can be changed by the Processor Controller microcode to any specified value within the adjustment range of the regulator (usually ±7% around nominal).
3. ***Monitoring of the regulator output*** signals is accomplished by an element of the DSS called the *tracking analog/digital and comparator* (TAD) subsystem (see Fig. 2). This subsystem provides continual monitoring of regulator voltages and currents against thresholds programmed into the TAD subsystem by means of the Processor Controller microcode. Each signal is represented in the subsystem by an eight-bit digital value. This value is updated every 12 microseconds to reflect the latest state of the signal. Associated with each TAD value are two additional eight-bit quantities—the high and low threshold values. These high and low thresholds are established by the Processor Controller microcode. The TAD hardware compares these thresholds against the current eight-bit value representing the tracked signal. Any excursion of the signal outside the programmed thresholds is interpreted as an analog event, and this results in an interrupt being generated by the DSS to the Processor Controller microcode. This interrupt requests the microcode to analyze the power/thermal hardware and, if the signal remains outside the programmed thresholds, to power-off the affected power partition.
4. ***Monitoring of TCM temperatures*** is also performed in a manner very similar to that for the regulators. The primary difference is that only a high threshold is monitored for TCM temperature.
5. ***Digital readout*** of voltages, currents, and coolant temperatures is provided by the *central analog/digital converter* (CADC). In fact, the CADC replaces the meter used by service personnel on predecessor systems and provides a simple and accurate means to display the critical voltages, currents, and temperatures of the system.
6. ***Interrupts*** from the power/thermal elements are presented to the Processor Controller via the DSS. These interrupts signal any abnormal event and initiate an analysis by means of the Processor Controller microcode.

Microcode structure

To provide the required microcode support in an efficient manner, common supervisor routines are used by all support functions. These routines, together with the hardware interrupt structure of the functional controller, provide for fast invocation of high-priority functions. During task switching, the execution of the lower-priority support function is suspended, with the associated code and data rolled out into the auxiliary storage adapter. This process is repeated as high-priority functions are invoked. Upon completion of the high-priority function, the code and data for the rolled-out functions are then rolled in and restarted. This task-switching capability is significant since the larger number of lower-priority Processor Controller functions, such as fault analysis, should not be permitted to lock out a higher-priority function such as error recovery or system reset.

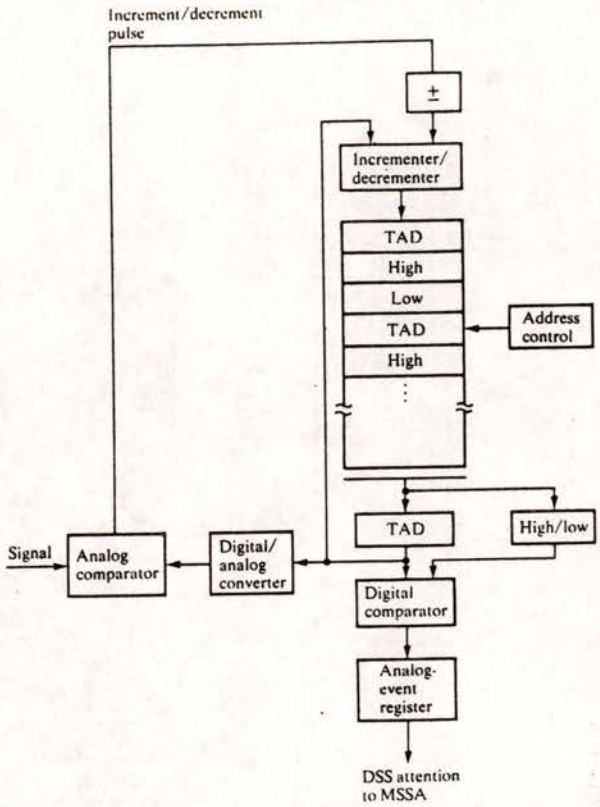


Figure 2 Tracking analog/digital and comparator (TAD) subsystem.

The console structure is designed to permit concurrent activities of operation and either local or remote maintenance. To ensure independence of this concurrent activity, a protection mechanism is built into the microcode which associates the hardware elements of the system with a particular console. Support microcode controlling the interface from the MSSA verifies the authorized association prior to issuing commands to the hardware elements. If the hardware is not associated with the console from which the request originated, the request is rejected and an error message is generated. This association is established during the configuration process, which is described in more detail subsequently.

Hardware support microcode structure

As opposed to building support functions (such as RESETs and DISPLAYs) into the LSI hardware of the system, and also to offset the lack of scoping capability, a primary

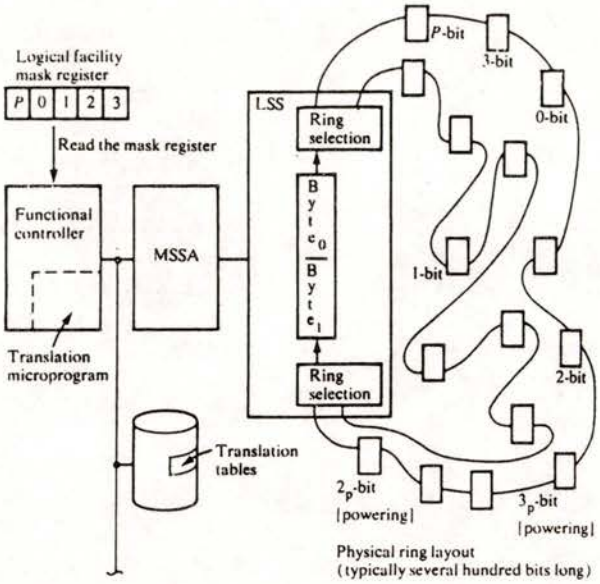


Figure 3 LSSD implementation.

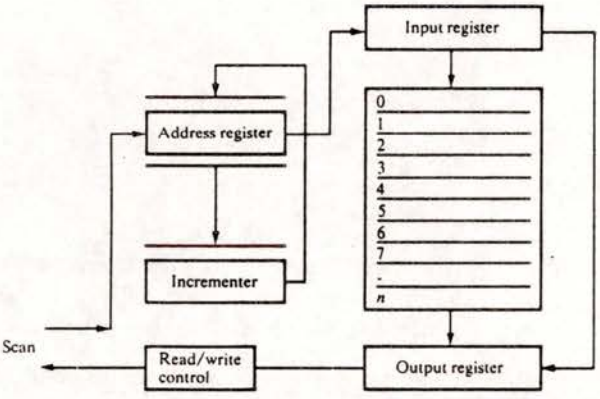


Figure 4 Array scanning with address incrementer.

objective was to provide these functions via the Processor Controller and the LSSD capability. This would provide a simpler hardware design with better overall flexibility. To accomplish this, a structure was established in the Processor Controller microcode to permit easy access of LSI facilities (*e.g.*, registers, arrays) by their symbolic names (*e.g.*, AREG). The relationship associating the symbolic names and the physical LSSD scan rings is held in tables to ensure independence from facility structure variations and engineering changes. The capabilities of this structure can be best understood with the aid of Fig. 3, which shows a typical LSSD implementation with two scan rings connected to a logic support station (LSS). The request to read a facility is received by the support structure, which then indexes into the translation tables to determine the scan rings and bit position that make up the facility. The data from the rings are scanned into the Processor Controller, and the facility is constructed per this table information. Correct facility parity, as well as powering (which is the replication of a facility for logic driving purposes), are also checked. When the operation is complete, the hardware is restored to its original state. On a write operation, only the requested facility is changed. The Processor Controller automatically determines and sets the correct state for the parity and powering bits associated with the facility. The preservation of unrelated data is accomplished by rotating the data in the scan ring, stopping the rotation to read or write selected bits, such that completion results in the ring being shifted to its original position.

The accessing of arrays is more involved and time-consuming. Figure 4 shows a typical array implementation. For a write-array operation, the write control, the address register, and the input register must be scanned. The array clocks are started for one cycle to write the input register into the array. To initialize an entire array, the following process is followed for every location of the array: scanning out of the address register, incrementing of the address, scanning of the address and input data back to the array, and cycling of the array clocks again. To improve the performance of this read operation, and for a write operation such as reset wherein the same data are stored into all locations of an array, a hardware address register incrementer was added to arrays such that, when the array clocks are cycled, the address is incremented by one. This hardware incrementer is shown in Fig. 4. The operation for writing the same data to consecutive locations can be a series of write-array clock commands. For a read operation, a series of scans followed by read-array clock commands are sufficient to collect data from all locations. This technique significantly reduces the total array write/read time and is used extensively by reset and logout functions.

Since this translation function required a large amount of table information stored on the Processor Controller file, the performance for certain functions (such as resets and diagnostics) was not acceptable. To resolve this, a compiler was developed which performs this translation process, yielding data for complete scan rings, thus permitting the bypassing of the translation function.

Hardware clock control microcode structure

The use of LSSD requires the stopping of system clocks to permit scanning. Clock control is provided in the LSSs such that clocks may be stopped on a per-component basis, thus permitting the scanning of a component while the remaining components are unaffected. To prevent spurious signals from

the TCM board being scanned, the components which are running are signaled, by the Processor Controller via the LSS, to degate all lines from the component being scanned. This capability permits as much concurrency between system operation and support function as possible. For example, recovery can run on a central processor component concurrently with system operation utilizing the remaining Processor Unit components.

Processor Controller storage path

A section of main storage is reserved by the Processor Controller for hardware support purposes. This *hardware system area* (HSA) is used to hold loadable microcode, control blocks, and trace data required or generated by the processor and channel hardware. A dedicated path to main storage is provided via the system controller such that access to main storage by the Processor Controller can be concurrent with system activity. This capability has been exploited to develop a real-time communication mechanism between the Processor Controller and the processors, channels, and *system control program* (SCP). Most important of these is the interface to the SCP, which is used to simplify the operation of the system, as is described subsequently.

Support functions

The support microcode structure, built on top of the hardware structure and paths into the system, provided an excellent base for the development of necessary application microcode. The structure provided all of the support function and system visibility that the applications required to initialize, control, recover, and maintain the system.

Initialization of the 3081, to the point where the SCP can be loaded, is controlled by the Processor Controller. Hardware controls in the power distribution frame and MSSA initialize the Processor Controller, which then powers on the remainder of the system. The regulator thresholds are established and the output of the regulator is measured by the Processor Controller microcode. This value is then compared with the expected nominal voltage for that regulator. If they are not equal, the regulator is adjusted and another measurement and comparison made. This measure-compare-adjust process is continued until the desired voltage is obtained. At this point, the tracking analog-to-digital value for that regulator represents the nominal output, and the high and low thresholds are established around this value. The temperature thresholds for each TCM have been predefined and are set at this time.

The LSI logic of the system is then reset using the LSS scan function. As part of the initialization process, the Processor Controller dynamically locates an error-free main-storage location for the assignment of the SCP *initial program load* (IPL) area, starting with address zero, as well as an error-free area to be used as the hardware system area. (See Fig. 5.) This is done by a processor microcode routine which validates main storage. Main-storage page locations containing a double-bit uncorrectable error, or that fail the pattern tests, are passed to the Processor Controller, which maintains a map of these failing pages. Assignment of storage is accomplished by the Processor Controller analyzing this map and initializing a storage-address-control array contained in the System Controller. The Processor Controller places this map of failing locations into the hardware system area in such a form that it can be referenced by the new TEST BLOCK instruction, which is utilized by the SCP to determine failing storage pages which must not be used [4].

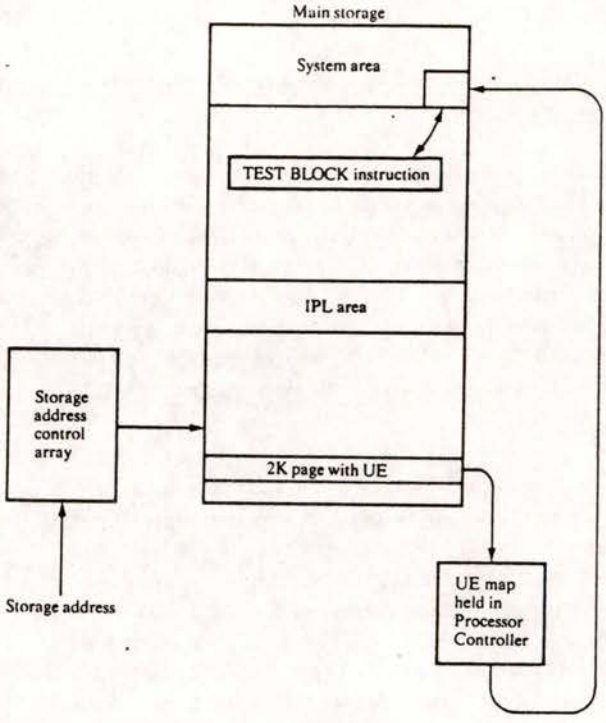


Figure 5 Storage configuration. (Note: UE = uncorrectable error.)

For *controlling* the system, prior to the IPL of the SCP, the operator selects desired configuration and control options by using the Processor Controller functions from the system console. The Processor Controller configures the hardware per the selected options and, at SCP IPL time, passes the configuration information to the SCP via the Processor Controller's interface to main storage [5]. Once the SCP has completed its initialization, configuration changes can only be accomplished by directing the request to the SCP. When the SCP has effected the logical part of the request, it signals the Processor Controller to perform the physical reconfiguration [6]. In the case of reconfiguring

storage, the Processor Controller, in conjunction with the SCP, moves data from one storage location to another to permit taking a storage unit off line for maintenance purposes.

This SCP-to-Processor-Controller communication is accomplished by a new processor instruction MSFCALL, an extension of the DIAGNOSE instruction. Execution of the instruction causes a processor to set up a control block in the hardware system area and signal the Processor Controller. The processor then continues instruction processing while the Processor Controller executes the request. Upon completion of the request, the Processor Controller signals the SCP via an external interrupt.

Exception handling for the 3081 system is performed by the Processor Controller. With the configuration established and the SCP running, the Processor Controller concentrates on monitoring the elements of the system, polling and displaying system status at the system console. In this mode, the Processor Controller is primarily waiting for exceptions, in the form of errors, to occur in the system. The objective is to handle these exceptions in a minimally disruptive fashion while collecting as much data as required for use in the automatic fault-isolation procedures which run on the Processor Controller. A technique used to minimize the effect of a hardware error is to stop only the clocks of the failing component, permitting the remaining components of the system to continue. If the error is an *uncorrectable error* (UE) in memory, the Processor Controller updates the storage error maps described earlier.

When a central processor encounters a solid error and is placed in the check-stop state by the Processor Controller, the store-in-cache design results in master data being locked in the cache of the processor. By accessing the System Controller, the Processor Controller can determine that this condition exists for specific storage addresses. The Processor Controller then issues a special command to the System Controller to force bad data for each address of main storage corresponding to this locked data [7]. In this manner, data integrity is maintained and, since most data contained in the cache at the time of error pertain to a unique task, only that task will be affected.

Another example of how the Processor Controller is utilized to improve system availability is in the recovery from storage errors caused by alpha particles [8]. On detected UEs, the Processor Controller, via a special algorithm, applies a test pattern to the data area to determine where the hard error exists. Through the algorithm, the soft error (alpha-particle-induced) is corrected and the data are restored. The central processor that detected the error is then retried successfully.

The power and thermal components of the system are continuously monitored and tracked. Exceeding the operating limits causes an interrupt to the Processor Controller, which logs all associated data for fault analysis and takes the least-disruptive action where this does not conflict with hardware protection. A key ability of this exception handling is the automatic correlation of a power/thermal exception to hardware errors that are likely to occur at essentially the same time. This capability is used to prevent the replacement of hardware when in fact the hardware error was caused by the power/thermal event.

Maintenance of the 3081 is a key role of the Processor Controller. To meet the maintenance time and cost objectives of the 3081, both error-detection and fault-diagnosing capabilities were enhanced. Error detection was stressed and evaluated early in the design cycle when improvement could be easily made [9]. In addition, the Processor Controller was exploited to perform thorough error data collection and analysis. In most cases, sufficient data are collected on the first occurrence of a fault to permit direct isolation to a very small number of *field-replaceable units* (FRUs). The correlation of multiple faults, both power/thermal and hardware, significantly enhances the Processor Controller's ability to isolate the problem to a small set of FRUs. In addition to fault analysis, a set of very comprehensive hardware tests can be run to isolate solid problems and to verify the repair action. These techniques are described in detail in the paper by Tendolkar and Swann in this issue [10].

Should these maintenance procedures fail to resolve the problem, the Processor Controller provides a large assortment of tools to be used by service personnel to identify and fix the problem. These tools capitalize on the LSSD and clock control functions for thorough visibility of the hardware. A remote capability exists which permits all Processor Controller functions to be used by highly trained personnel in a support center.

In cases where the Processor Controller hardware contains a solid fault, built-in diagnostics and analysis routines are utilized. In this case, the support personnel must perform parts replacement using a four-digit reference code indicator on the system support panel (Fig. 1). Should this repair be unsuccessful, the Maintenance Device [11] is attached to a receptacle provided in the Processor Controller. The support personnel are led through a question-and-answer dialogue, at the Maintenance Device keyboard, intended to find and fix all residual hardware failures not found by the built-in diagnostics.

Conclusion

The Processor Controller represents a significant evolution in meeting the needs of a large-scale LSI computer system.

It has extended the role of processor controllers on predecessor systems in key areas, offsetting potential problems associated with LSI hardware. It has provided traditional functions, such as resets, in a cost-efficient fashion, reducing the number of circuits required in the 3081 by exploiting LSSD to provide these functions. In addition, the flexibility it provides in these areas has contributed to efficient hardware bring-up and to reducing hardware change activity in the IBM 3081 Processor Complex. The Processor Controller design has also been successfully exploited to attain very high objectives in the area of improved system availability and operational characteristics. By providing improved clock-control granularity, automatic configuration control, and automatic fault isolation, system incidents and outages due to hardware failures or operator errors should be reduced.

Acknowledgments

The authors wish to recognize the significant contributions to the Processor Controller development effort made by G. D. Granito and J. D. Vowell, who established many of the key objectives and who were persistent in their efforts to ensure that those objectives were met. Acknowledging the efforts of the many engineers responsible for the design effort would be difficult. However, the authors wish to recognize a few who made significant contributions to the design direction: T. A. Stranko for the hardware design, L. A. Guadagno for the power/thermal structure, E. Borchsenius for the LSI service tools, and C. L. Baldwin and S. P. Geiger for the structure of the Processor Controller microcode.

References

1. M. S. Pittler, D. M. Powers, and D. L. Schnabel, "System Development and Technology Aspects of the IBM 3081 Processor Complex," *IBM J. Res. Develop.* **26,** 2–11 (1982, this issue).
2. R. M. Gustafson and F. J. Sparacio, "IBM 3081 Processor Unit: Design Considerations and Design Process," *IBM J. Res. Develop.* **26,** 12–21 (1982, this issue).
3. E. B. Eichelberger, "Level Sensitive Logic System," U. S. Patent 3,783,254, 1974.
4. E. L. Richardson, R. M. Smith, A. J. Sutton, and J. D. Vowell, "Retention of Failing Storage and Storage Protect Key Address Through Initial Program Loads," *IBM Tech. Disclosure Bull.* **22,** No. 1 (January 1980).
5. B. B. Moore, J. T. Rodell, A. J. Sutton, J. D. Vowell, and P. J. Wanish, "System Control Program Initialization in a Multiprocessing System," *IBM Tech. Disclosure Bull.* **22,** No. 9 (February 1980).
6. B. B. Moore, J. T. Rodell, A. J. Sutton, and J. D. Vowell, "Automatic Vary of a Central Processor in a Multiprocessor System with Non-Store-Through Cache," *IBM Tech. Disclosure Bull.* **22,** No. 4 (September 1979).
7. E. J. Annunziata, B. L. McGilvray, and A. J. Sutton, "Cache Purge in Store-in-Cache System," *IBM Tech. Disclosure Bull.* **20,** No. 9 (February 1978).
8. D. C. Bossen and M. Y. Hsiao, "A System Solution to the Memory Soft Error Problem," *IBM J. Res. Develop.* **24,** 390–397 (1980).
9. D. C. Bossen and M. Y. Hsiao, "Model for Transient and Permanent Error-Detection and Fault-Isolation Coverage," *IBM J. Res. Develop.* **26,** 67–77 (1982, this issue).
10. Nandakumar N. Tendolkar and Robert L. Swann, "Automated Diagnostic Methodology for the IBM 3081 Processor Complex," *IBM J. Res. Develop.* **26,** 78–88 (1982, this issue).
11. *Processor Controller Maintenance Manual,* Order No. SY22-7063, IBM Corporation; available through IBM branch offices.

Received June 9, 1980; revised July 6, 1981

The authors are located at the IBM Data Systems Division laboratory, Poughkeepsie, New York 12602.

Nandakumar N. Tendolkar
Robert L. Swann

Automated Diagnostic Methodology for the IBM 3081 Processor Complex

The concepts of automated diagnostics that were developed for and that are implemented in the IBM 3081 Processor Complex are presented in this paper. Significant features of the 3081 diagnostics methodology are the capability to isolate intermittent as well as solid hardware failures, and the automatic isolation of a failure to the failing field-replaceable unit (FRU) in a high percentage of the cases. These features, which permit a considerable reduction in the time to repair a failure as compared to previous systems, are achieved by designing a machine which has a very high level of error-detection capability as well as special functions to facilitate fault isolation using Level-Sensitive Scan Design (LSSD), and which includes a Processor Controller to implement diagnostic microprograms. Intermittent failures are isolated by analyzing data captured at the detection of the error, and the analysis is concurrent with customer operations if the error is recoverable. A further improvement in the degree of isolation is achieved for solid failures by using automatically generated validation tests which detect and isolate stuck faults in the logic. The diagnostic package was designed to meet a specified value of isolation effectiveness, stated as the average number of FRUs replaced per failure. The technique used to estimate the isolation effectiveness of the diagnostic package and to evaluate proposals for improving isolation is described. Testing of the diagnostic package by hardware bugging indicates very good correlation between projected and measured effectiveness.

Introduction

A major portion of the logic of the IBM 3081 Processor Complex is implemented using *level-sensitive scan design* methodology [1–3]. Up to 118 LSI chips are mounted on a *thermal conduction module* (TCM) [4] which is the *field-replaceable unit* (FRU). A 3081 Processor Unit with 16 channels contains 26 TCMs and one with 24 channels contains 27 TCMs. The subject of this paper is the design of diagnostic microprograms that are capable of isolating failures if they occur in these TCMs, on the boards on which the TCMs are packaged, or in the interboard cables. A major design goal was to isolate automatically all hardware-detected errors caused by hardware failures. This goal has the following advantages: 1) the duration of repair action is reduced because the set of FRUs required to repair the 3081 is identified by the diagnostic microprograms, which operate at microprocessor speed, and is communicated to the *customer engineer* (CE), who can bring these FRUs with him; 2) the level of training required by a CE to repair the 3081 is reduced, since the CE need not know the internal logic; and 3) the same reliable service is available to all customers, since the CE's skill in diagnosing a failure is not a factor. Another design goal for the diagnostic package is that it be able to isolate, to a single TCM or to a set of TCMs, both intermittent and solid (permanent) failures for a specified percentage of failures. The word *isolation* is used here to imply that the diagnostics will identify (call) a set of TCMs such that the failing TCM is in the set. If the set contains no other TCM than the failing one, the isolation is unambiguous and the 3081 can be repaired by replacing the identified TCM. If the set contains more than one TCM, the 3081 can be repaired by replacing at most all TCMs in the set.

A figure of merit to compare different diagnostic strategies is the *isolation effectiveness*, defined here as the average number of FRUs called by diagnostics for a failure. This is an important figure of merit because the cost of the FRUs replaced in the field is directly proportional to the isolation effectiveness. The actual isolation effectiveness is a function of several factors including the ratio of intermittent to solid

Copyright 1982 by International Business Machines Corporation. Copying is permitted without payment of royalty provided that (1) each reproduction is done without alteration and (2) the *Journal* reference and IBM copyright notice are included on the first page. The title and abstract may be used without further permission in computer-based and other information-service systems. Permission to *republish* other excerpts should be obtained from the Editor.

faults (we describe this later in the section on diagnostic effectiveness). Although the goal is to identify a failure to the failing FRU unambiguously, the designers of the diagnostics for the 3081 had to design a system that has an isolation effectiveness of 1.2 TCMs per failure or less.

There is a major difference between the technique used for diagnostics in the 3081 and that used in existing systems [5–7], which mainly rely on being able to recreate a failure by running diagnostic tests on the failed machine. In the 3081, a new approach, which uses the data captured at the detection of the error, is used to isolate a failure. Advantages of this method over the former for isolating intermittent failures are discussed in the paper by Bossen and Hsiao [8] in this issue.

This paper describes the techniques used to achieve the previously discussed goals and the evolution of the automated diagnostic methodology. This is followed by a review of the key features of the 3081 Processor Complex hardware that facilitate the implementation of automated diagnostics. A brief description of the Processor Controller's role in supporting diagnostics is given; the fault-isolation process and the four algorithms that form the heart of the methodology are described; and techniques for determining the average number of TCMs-called-per-failure are presented. Highlights of the key features of the automated diagnostic methodology are discussed in the concluding section. The fault-isolation process applies to diagnostics of all failures of the TCMs, the boards on which TCMs are mounted, and the interboard cables, which constitute a major portion of the 3081 hardware.

Overview of automated diagnostic methodology

A primary requirement for designing a diagnostic methodology that automatically isolates failures to a set of FRUs is that some hardware mechanism should detect a very high percentage of failures. Since failures cause errors and since well-known techniques are available to detect errors [9], the IBM 3081 Processor Unit is designed with extensive error-detection mechanisms. It has a capability of instantaneously detecting an estimated 90% of the errors caused by hardware failures; this estimate is based upon the definition and methodology described in the paper by Bossen and Hsiao [8].

The next major requirement is to design a scheme for isolating the detected failures to meet the average-FRUs-per-failure goals for diagnostics. Once an error is detected, certain information that describes the state of the machine at the time of the occurrence of the error is collected. This information determines the *isolation effectiveness* (average number of TCMs called by the diagnostics per failure). A failure is considered to be isolated to a set of FRUs if the detected error could have been caused by failure of some hardware on one of the FRUs in the set and not by the failure of any other FRU not in the set. In order to fix the failure with certainty in a single repair action, all FRUs in the set must be replaced. To meet the stated goals for isolation effectiveness, the hardware should be designed so that the information required to isolate the failure must be available to the diagnostic microprogram. Finally, a given physical failure may cause a set of errors, possibly with distinct symptoms, from which the system can recover. The total information collected for the entire set of errors may result in further isolation of the failure. Thus, a set of algorithms is required to process the information collected for a set of errors to achieve the best possible isolation.

These algorithms isolate the failure to a failing FRU in many cases, but there is still a class of failures for which the isolation is greater than one FRU. For those cases, two additional techniques are used for further isolation. Using the LSSD facilities, the suspected TCMs can be tested for the existence of *stuck-at* faults (*i.e.*, logic stuck at either 1 or 0 levels). If any TCM has a stuck-at fault, it is identified by these tests. A certain percentage of failures do not fall into the stuck-fault category. For such failures, these algorithms do not identify the single FRU that has failed; we just know the set of FRUs, one of which has failed. A procedure called *sequential FRU replacement* (SFR) has been developed and implemented to reduce the average number of TCMs called by diagnostics to fix such failures. The SFR policy identifies a subset of the set of FRUs such that the failing FRU is in the subset with a very high average probability. These FRUs are replaced in the first repair action when the failure occurs. In a very small percentage of the cases where SFR is used, a second repair action may be required. The implementation of this scheme is described subsequently.

Throughout the development of the diagnostics, it was necessary to evaluate whether they met the stated isolation objectives. The method used for evaluation, described subsequently, was also used to project what percentage improvement would occur in the isolation effectiveness if a proposed hardware or diagnostic algorithm change were to be implemented.

3081 Processor Complex hardware

Hardware features which are most significant to the diagnostics are that 1) the hardware has a very high level of error-detection capability, 2) the concept of the *active source identifier* (described subsequently) is developed and implemented to improve isolation, 3) information required to isolate a failure is stored in the LSSD *shift-register latches* (designated herein as SRLs) and arrays, and 4) the *monitoring and system support adapter* (MSSA) and the *logic support station* (LSS) provide a means for the Processor

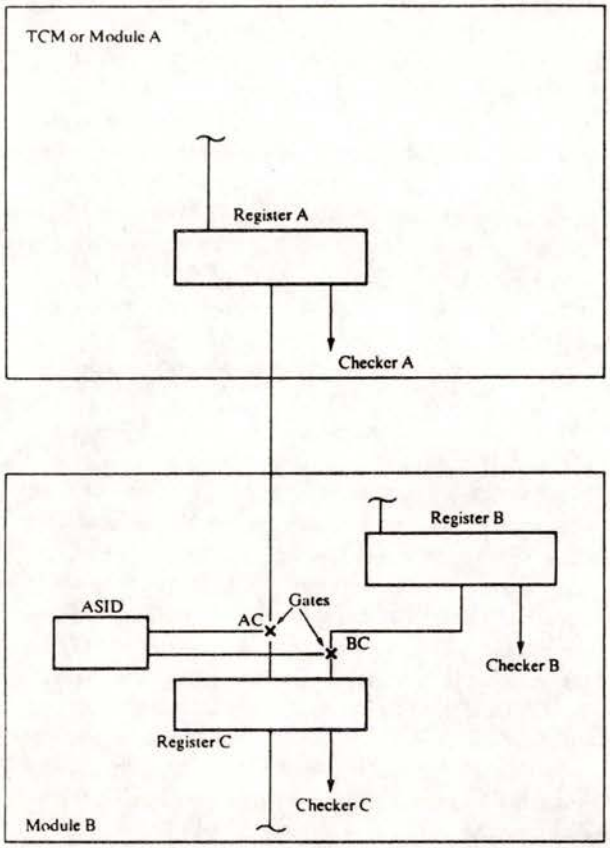


Figure 1 An example of the Active Source Identifier (ASID) concept.

Controller to know about the detection of an error and the ability to scan out the SRLs and arrays that contain the information required for isolation. (Note: characteristics of the Processor Controller are discussed in the paper by Reilly *et al.* [10] in this issue.) As a result of such features, diagnostic microprograms which run on the Processor Controller have the necessary information to isolate a failing TCM.

Error-detection mechanisms are extensively used in the 3081 Processor Complex. Checking techniques are used for hardware facilities such as registers, array chips, adders, decoders, and control mechanisms. When a checker detects an error, certain events are triggered which ultimately lead to the invocation of the diagnostic microprograms. We return to this later.

The concept of the *active source identifier* is illustrated in Fig. 1. Suppose Checker C detects an error whereas Checkers A and B do not indicate any errors. If we did not know whether Register B was ingated or Register A was ingated at the time of the error, then the failure could be either on Module A or on Module B. Two FRUs would have to be called for this detected error. Let us suppose that an SRL (labeled ASID) is used to indicate whether Gate AC or BC was open when the data were transferred into Register C. Then, when a failure is detected while transferring data from Register B to Register C, the ASID would indicate Gate BC, and the failure would be isolated to Module B. Thus the ASID helps in reducing the number of FRUs called by diagnostics where one or another path could have been used at the time of the failure. In general, the *active source identifiers* are SRLs or arrays which identify the data source or the control function which was active at the time of the error. They are used in places where the error-detection mechanism can be set by failure on more than one source module.

Arrays have been provided to store the information required to isolate a failure. Since it takes several cycles before the clocks of a given portion of the hardware (referred to as a *functional element*) that has detected an error can be stopped, the arrays contain several cycles' worth of values of the active source identifiers and control words. The error checkers are SRLs that, once set to *on*, cannot be reset by any mechanism other than the *logic support station* (LSS). The LSS provides a mechanism to stop the functional element that has detected an error. It is connected to the processor controller via the MSSA. The LSS presents an interrupt to the processor controller to indicate that an error is detected.

Role of the Processor Controller

The complete functions of the Processor Controller are discussed in the article by Reilly *et al.* [10]. The relationship of the Processor Controller to other units of the 3081 Processor Complex is shown in Fig. 2. The diagnostic microprograms and the error handler and recovery microprograms run on the Processor Controller. It also supports power and thermal error detection and failure isolation; thus, logic errors due to failure of power or cooling equipment can be separated from those due to failure of TCMs.

When the 3081 hardware detects an error, the LSS presents an interrupt to the Processor Controller and also stops the clocks of the functional element involved. The error handler and recovery microprogram collects the information that is required for both fault isolation and recovery by scanning out the SRLs and arrays. The error handler tries to recover the operation and, if successful, the 3081 can continue data processing operations. The error handler records the scanned-out data on the *Processor Controller File* (PCF). This recording is called a *logout*. The error handler then posts the diagnostics with a message that an error has been detected and passes a pointer to the location of the logout. The diagnostic microprograms use this logout to determine the failing TCM.

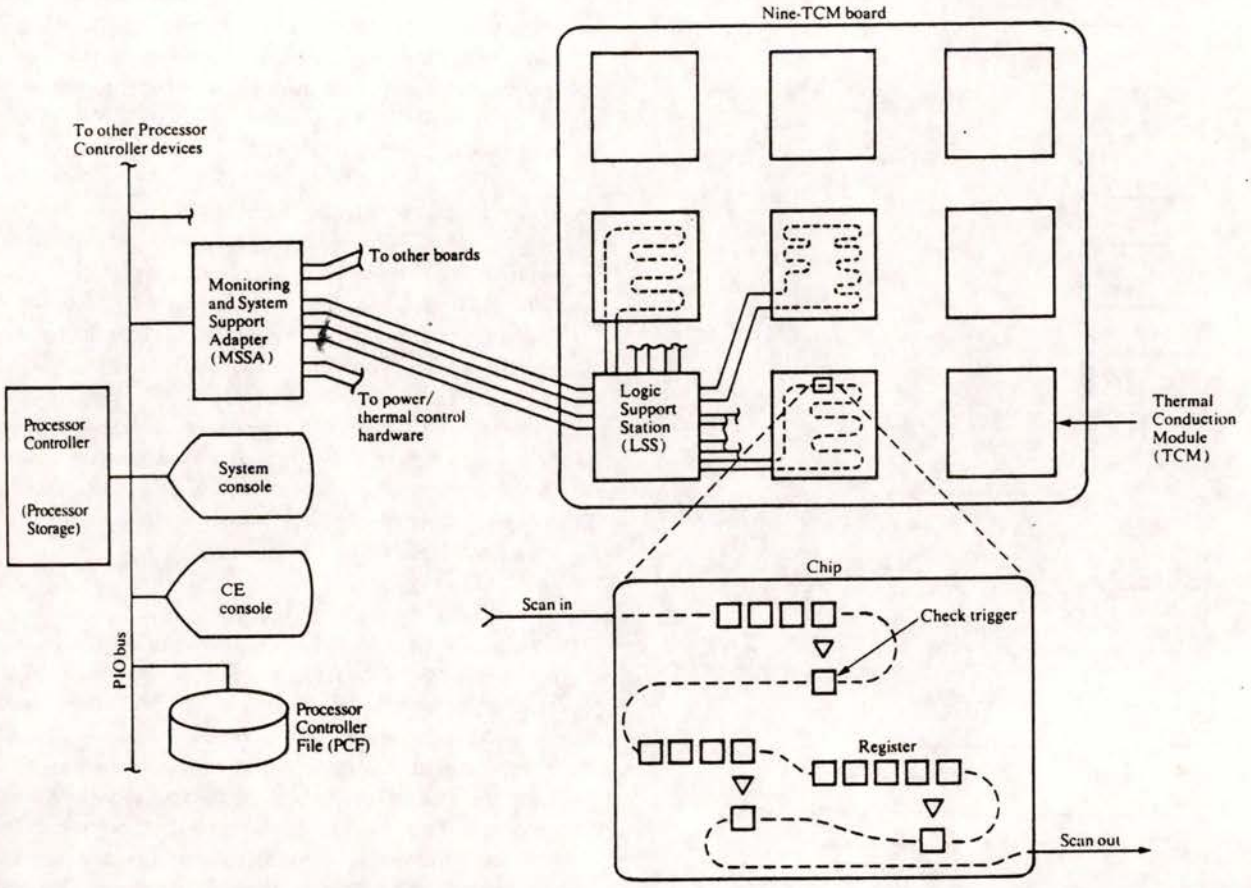


Figure 2 Block diagram illustrating relationship of the Processor Controller to other units of the 3081 Processor Complex.

The Processor Controller also provides facilities for the diagnostic microprograms to communicate to the *customer engineer* (CE). A display console is provided for the CE to obtain information about any detected failure and the failing TCMs identified by diagnostics for each failure. If the frequency of errors is higher than a threshold value or if a processor is in check-stop state, the diagnostic microprogram also displays a code on the system console indicating which TCMs the CE needs to bring with him to fix the problem.

Fault-isolation process

The function of the fault-isolation process is to determine a minimal set of FRUs that must be replaced to fix a failure or a fault that has caused an error detected by the hardware. There are four elements that make up the fault-isolation process: *direct isolation, intersection isolation, validation tests* (VTs), and *sequential FRU replacement* (SFR). The fault-isolation process is illustrated in Fig. 3. Whenever an error is detected by the 3081 hardware, the error handler makes a logout and invokes diagnostic microprograms. The logouts are processed by *fault-isolation analysis routines* (FIARs). This phase corresponds to direct isolation. If the result of direct isolation is a multiple FRU call, intersection isolation is invoked to further isolate the failure. If the frequency of the error is over a threshold (currently set at 24 errors in any continuous two-hour period) or if a processor is put in check-stop state because it is impossible or undesirable to continue operation, a message is sent to the system console indicating that a CE be called, and a coded number is displayed which is used by the CE to bring the necessary set of TCMs.

The diagnostic microprograms maintain a file that contains, for each detected fault, a block of information called a *fault-isolation record control block* (FIRCB). The CE can look at an FIRCB and determine the FRUs called by diagnostics and any further processing that may be required. If direct isolation and intersection do not result in

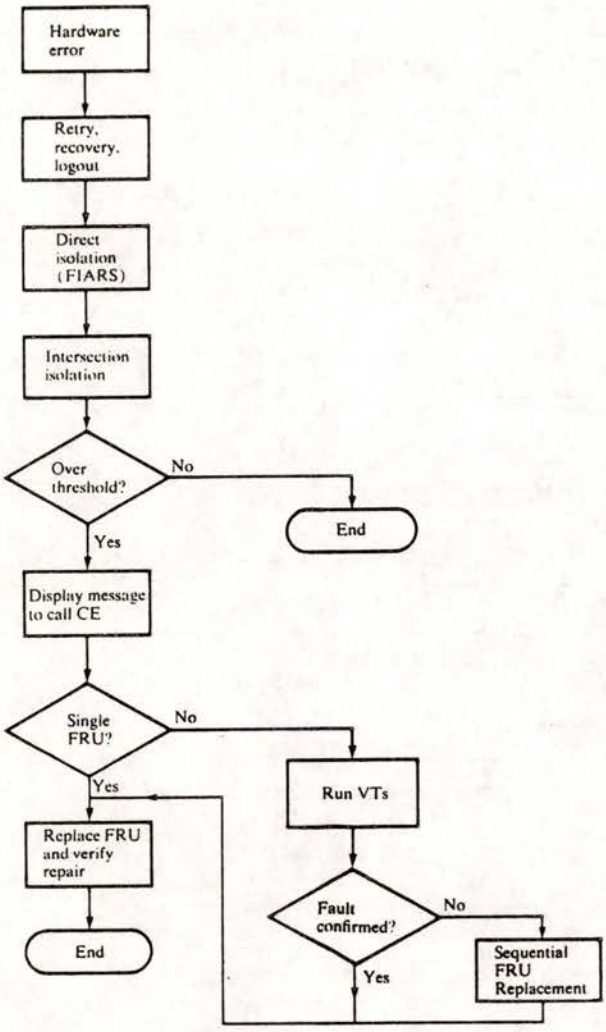


Figure 3 Fault-isolation process.

isolation to a single TCM, the diagnostic microprogram indicates (in the FIRCB) to the CE that the system operator should release the indicated hardware to run VTs.

The CE comes to repair the machine either because the diagnostic microprograms detected an over-threshold condition or the customer determined that the rate of errors is unacceptable and the machine needs repair. If the fault was isolated by direct or intersection isolation to a single TCM, the CE proceeds to replace the indicated TCM. If it was not isolated to a single TCM, the CE makes the necessary hardware available to diagnostics (with the customer's permission) for running VTs. If the VTs detect a fault and identify the failing TCM or TCMs to be replaced, the CE replaces them. If the fault was not detected by the VTs, the SFR policy is invoked and the set of TCMs to be replaced in the first call are conveyed to the CE, who replaces them. The last step in the process is verification of the replacement. In this step, we ensure that there are no stuck faults remaining in the system after replacement by running VTs to detect such faults.

After replacement, the diagnostic file contains information on which TCMs were replaced and when. If the fault recurs in the two-week period following a repair, one of the following applies: sequential FRU replacement policy was used and the second call is required; the failure may be on a board or a cable if the domain of the error so indicates; or all indicated FRUs, including boards and cables, have been replaced and the error still persists. In the first case, the CE proceeds to replace TCMs indicated in the second SFR call. In the second and third cases, the CE calls a support center for help because further analysis is required, in either case, before any repair action can be taken.

- *Direct isolation*

Direct isolation is defined as the process of isolating a failure based only on the logout information collected by the error handler for a single detected error. The isolation of failures from the logout would be a complicated, if not infeasible, process if an unknown number of physically distinct failures could simultaneously occur. If the failures occur as a random process, and if the failure rate is such that the probability of a second failure occurring before the first failure is repaired is very small (1% or less), a single fault will be present 99% or more of the time when an error is detected. For the technology used in the 3081 Processor Unit, projection of failure rates based on testing in the laboratory environment indicates that this condition is satisfied and the single-fault assumption is justified.

An important function of the direct-isolation process is also to create information used by the other elements of the fault-isolation process. We now introduce the concept of a *direct-isolation domain* (DID), which is a collection of hardware entities, and the concept of error syndromes. We show how a relationship is established between an error syndrome and a direct-isolation domain. The process of direct isolation consists of examining the logout to determine which error syndrome has occurred and, using the relationship between error syndromes and DIDs, determining the DID that contains the failure.

Error syndromes A *syndrome* is a specific combination of values of the bits in the logout. The logout consists of a certain number of bits of information on the state of the machine at the time of the detection of the error. Included is information on the status (*on* or *off*) of the error checkers, active source identifiers, and other information stored in the arrays for fault-isolation purposes (*e.g.*, what function was being performed, etc.).

Direct isolation domain (DID) of a syndrome The DID of a syndrome is a set of hardware entities (SRLs, registers, encoders, decoders, or some collection of circuits) such that when any of these fail, they could cause the error syndrome to occur, and further, there is no other hardware entity outside the set whose failure could cause the same syndrome. Clearly, the definition of the DID of a syndrome implies that the error syndrome could be caused by the failure of only those FRUs on which the DID of the syndrome lies. Thus, the problem of isolating the failure requires finding the DID of the syndrome.

Consider a network of interconnected circuits that is a subset of the 3081 Processor Unit hardware. Furthermore, this network has a fixed number of inputs and outputs. Let the set of inputs be such that for a given syndrome there is no error in the inputs, and the set of outputs be such that it corresponds to the error syndrome detected. In this network, the output is a function of the inputs and the logic of the network. Since inputs are error-free and an output has errors in it (if there is a single fault in the system), the fault is in the set of circuits in this network. If we can find a network that satisfies these conditions, but no proper subset of it satisfies them, the set of circuits in the network is the DID of the syndrome. DIDs are determined by back-tracking algorithms and are verified by simulation. Using the previously described process, a DID is determined for each valid error syndrome by the diagnostician. An example of a DID is shown in Fig. 4.

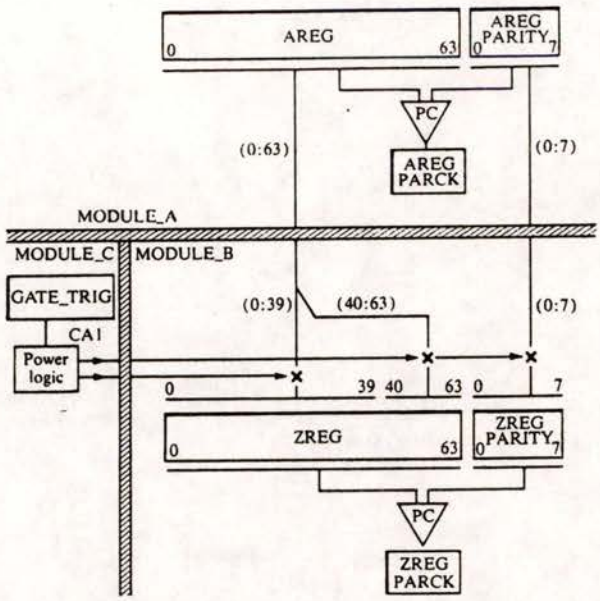
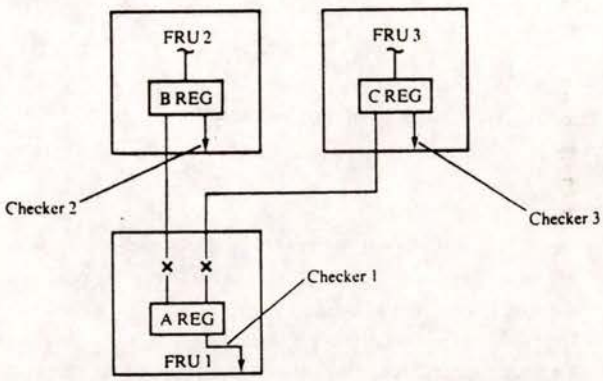


Figure 4 Example of a Direct-Isolation Domain (DID). The syndrome is ZREG PARCK *on* and AREG PARCK *off*.

• *Intersection isolation*

Intersection isolation is the technique of isolating a failure to a single FRU when the failure results in a variety of error syndromes over a period of time. An examination of the hardware of the 3081 Processor Unit indicated that there are hardware entities which, when they fail, can cause many different syndromes to occur depending upon the function being performed by the machine. Direct isolation would produce a DID for each such syndrome that occurs. Advantage can be taken of the collective information provided by several syndromes to further isolate the failure beyond what was accomplished by direct isolation.

Principle of intersection isolation When a group of errors occurs in a small time interval such that it is reasonable to assume that a single fault exists, the fault must be in the hardware area common to (*i.e.*, the logical intersection of) all error domains of the group. If this hardware area is on a single FRU, the fault is isolated to a single FRU for all errors of the group. This principle is illustrated by an example shown in Fig. 5. Error 1 occurs when data flow is from B REG to A REG. The domain of Error 1 contains the A REG on FRU 1 and the B REG on FRU 2. Sometime later, Error 2 occurs when the data flow is from the C REG to the A REG. The domain of Error 2 contains the A REG on FRU 1 and the C REG on FRU 3. Applying the principle of intersection isolation, it is concluded that the failure is on FRU 1, A REG, which is the area common to both domains.



Error	*Detected by checker*	*Domain* (FRU, area)	*Data flow*
1	1	(1, A REG) and (2, B REG)	B REG to A REG
2	1	(1, A REG) and (3, C REG)	C REG to A REG

Figure 5 Intersection isolation example.

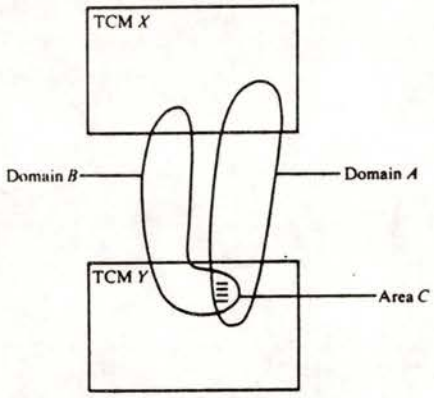


Figure 6 Intersection using TCMs.

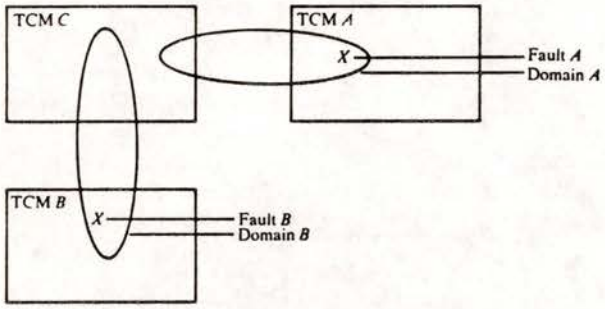


Figure 7 Intersection when two faults exist.

Hence, FRU 1 is the failing FRU. In the example shown in Fig. 5, the number of FRUs required to correct the problem is reduced from three to one.

To apply intersection isolation, we first have to identify the hardware facilities common to a set of (direct isolation) domains of a set of error syndromes. These hardware facilities could be latches, triggers, registers, etc. Since the hardware facilities in each domain are specified for the machine, one can determine the hardware facilities common to any set of domains. In the following discussion, the term *area* is used to denote a collection of hardware facilities such as latches, triggers, registers, etc.

Intersection isolation requirements The domains have to be defined in such a way that the area common to various domains can be identified; this requires the following:

1. Each FRU is divided into areas, and each area should meet the following criteria: a. No two areas have any hardware in common; b. The area lies entirely on a FRU; c. The area is either completely inside or outside a domain.
2. There is a domain D_i associated with each error syndrome i. This D_i is identified by direct isolation. For a given D_i a set S_i is defined as follows:

   $S_i = \{(M, A) \mid (M, A) \in D_i\}$,

   where $(M, A) \in D_i$ means that Area A on FRU M is a part of the domain of error i.

Intersection-isolation technique Given a set of error syndromes and associated domains, we define a set S^* as follows:

$S^* = \{(M, A) \mid (M, A) \in S_i$ for each error $i\}$.

The following three cases can arise:

1. If there is a FRU M^* such that for all $(M, A) \in S^*$, $M = M^*$, then the fault is isolated to FRU M^*.
2. If there are at least two distinct FRUs M_1 and M_2 such that $(M_1, A_1) \in S^*$ and $(M_2, A_2) \in S^*$, then the fault is not isolated to a single FRU.
3. If S^* is empty, then at least two failures (faults) exist in the system. This case can be converted into Cases 1 or 2 by partitioning errors into groups. A group is a set of errors such that there is at least one (M, A) that belongs to the domain of every error in the set. (S^* is not empty for all errors that belong to a group.)

Implementation of intersection isolation This technique would require a process to divide DIDs into areas. Because of the large number of DIDs involved in the 3081 and the lack of an automated process, an alternative way of using just the TCMs for intersection was investigated. Thus, if S_1, S_2, $\cdots$, S_n are the sets of TCMs corresponding to the domains of errors E_1, E_2, $\cdots$, E_n detected in a time period where single fault assumption applies, then S^*, where $S^* = S_1 \cap S_2 \cap \cdots \cap S_n$, is the set of TCMs to which the failure is isolated.

If there exists a single fault in the machine, the failing FRU is still in the intersection of the FRUs called for each DID, but there is a possibility that intersection using TCMs would not result in as good an isolation as obtained by using areas. An example of this is shown in Fig. 6. For both Domain A and Domain B, TCMs X and Y are called. The intersection of these two domains on a TCM basis still gives TCMs X and Y. If the failure was in Area C, it would not be isolated to TCM Y if just TCMs are used for intersection. The second concern is that there is a small probability that two distinct faults could occur which would result in a wrong (not failing) TCM called by intersection. This is shown in Fig. 7. There are two faults, one on TCM B and another on TCM A. The TCM C is the common TCM for their domains. The intersection would result in calling TCM C.

An evaluation of the impact of these two concerns on the isolation effectiveness was performed using random selection

of failure locations. The study showed that, based on the projected failure rates of TCMs, if we allowed only syndromes occurring within a two-week period of each other to intersect, the average number of TCMs per failure using TCMs for intersection would be 3% higher than that obtained by using areas. The major effort required to implement intersection using areas was not justifiable, so we decided to intersect the set of TCMs called by DIDs to isolate a failure. This means that when a set of DIDs occur in a two-week time period, the failure is isolated to TCMs common to each DID that occurs. To enable us to take corrective action in those rare cases where intersection does not call the correct set of TCMs, a record is kept of DIDs that were intersected for a given fault. If the error recurs, the identity of TCMs corresponding to the particular DID that recurred are still available to fix the problem.

• *Validation tests*

Validation tests (VTs) are tests for detecting and isolating stuck faults in combinational networks. Their generation and application in fault isolation for systems using LSSD has been extensively reported. References [11–13] contain techniques of generating tests for logic networks. References [1–3, 14] describe the concepts of LSSD and show how tests can be generated to isolate "stuck-at" faults for a machine designed with LSSD. For the 3081, validation tests are generated using the techniques described in these references. We now describe the role of validation tests in the overall diagnostic strategy and give some of the characteristics that are unique to the 3081.

The role of validation tests as part of the diagnostic microprograms is to confirm the existence of a fault in a TCM and, if a fault is found, to determine the failing TCM. The validation tests are also used to verify the repair action taken by the CE. VTs are primarily used for isolating those faults for which direct isolation and intersection isolation call more than one TCM. When the CE is called to fix such a problem, he is given instructions to vary off line the portion of hardware on which the VTs need to run. The CE signals the availability of the hardware, and the diagnostic microprograms automatically run the appropriate set of VTs. If the VTs find a fault, the names of the failing FRU or FRUs are displayed on the CE console and the CE is asked to replace them. If VTs do not detect a fault, sequential FRU replacement (described in the next section) is used to call FRUs.

Characteristics of validation tests Validation tests are part of the diagnostic microprograms which reside on the Processor Controller File. They are generated automatically by programs from the description of the logic of the 3081. The processes of scheduling the tests to be run, running the tests when hardware becomes available, and calling the failing FRUs are also automated.

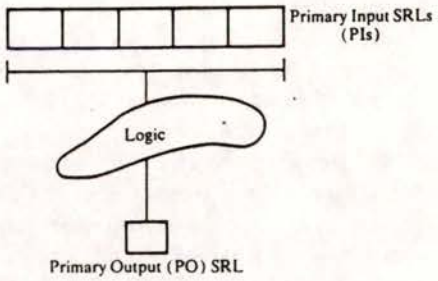


Figure 8 Segment of an SRL (shift-register latch).

The logic of the entire machine is divided into segments. A segment is defined for each SRL, called the *primary output* (PO) of the segment, in the machine. It consists of all SRLs such that the value of the PO SRL of interest at the end of this machine cycle depends on the current value of this set of SRLs [called *primary inputs* (PIs)] plus the combinational network whose output is the PO SRL. This relationship is shown in Fig. 8. A stuck fault in a segment can be detected by scanning a particular set of bits into the SRLs that are PIs and POs, running the machine one clock cycle, and comparing the value of the PO SRL to that of the same SRL of a fault-free machine. If the two values do not agree, there is a fault in the segment.

The running of VTs requires the scanning of data (test pattern) into a group of SRLs, running the clocks for one cycle, and scanning out a group of SRLs to detect a fault. A number of patterns are required to detect all possible sets of faults, and the procedure of scan-in, scan-out, and compare is repeated until a fault is found or all test patterns are exhausted and no fault detected. The latter case is assumed to be an intermittent fault. The scan-in, scan-out, and running of clock for one cycle is controlled by the Processor Controller. To minimize the time required to detect and isolate a given fault when multi-FRU DIDs are present, the entire set of tests is divided into groups of tests called a VT. Each has a unique name and there is a group of segments that the VT covers or finds faults in. Thus, only those tests that cover the suspected portion of the hardware need be run to isolate a given fault.

• *Sequential FRU replacement policy*

It was pointed out earlier that, for certain sets of failures, the direct and intersection isolation and validation tests cannot pinpoint the failing TCM (FRU). The DIDs of such failures are spread over more than one TCM. It is possible to fix such a failure with more than one repair action using, on an average, fewer replacement TCMs per failure than one

repair action would require. Sequential FRU replacement (SFR) policy applies to this set of failures, and it minimizes the average number of TCMs replaced per failure for a given average probability of fixing the failure in one repair action. A given failure is fixed in at most two repair actions. For a slight risk of not fixing a problem in the first repair action, a considerable reduction in the average number of TCMs per failure is achieved. From the point of minimizing impact to customer operations it was assumed that the average risk of requiring a second repair action for a failure in an SFR case should not exceed a very small value.

There are two aspects of SFR policy: the first problem is to determine the optimal set of TCMs to be replaced for a given failure to meet the objectives stated earlier, and the second problem is to implement the optimal strategy on the processor controller.

Optimal SFR policy An SFR strategy is completely specified by defining for each DID which TCMs are to be replaced in the first (repair action) call and which are to be replaced in the second. Corresponding to each strategy there is an average number of TCMs replaced per failure and a risk of second call. A computational procedure was developed by Rutledge [15] to determine the replacement strategy that has the minimum average TCMs per failure and for which the average risk of second call does not exceed a given percentage value. To calculate the average TCMs per failure and risk of second call, we use the following definitions. There are k DIDs in the system that call two or more TCMs. The failure rate of DID_i is the number of failures of circuits of DID_i in some specified time period T. It is proportional to the number of circuits, f_i, in the DID. Let Q_i be the conditional probability that, given there is a failure in the hardware covered by these DIDs, it is in DID_i. Then

$$Q_i = \frac{f_i}{\sum_{i=1}^{k} f_i}.$$

If DID_i has circuits on N_i TCMs, then $P_{j/i}$ is the likelihood that, given that a failure exists in DID_i, it is on TCM_j. $P_{j/i}$ is given by the following equation:

$$P_{j/i} = \frac{\text{Number of circuits of DID}_i \text{ on TCM}_j}{\text{Total number of circuits in DID}_i}.$$

For a given DID, for any strategy, TCMs are replaced in the order of decreasing likelihood. This reduces the average number of TCMs per failure and the risk of second call for a fixed number of TCMs that are replaced in the first call. If a failure occurs in DID_i, we replace TCM_j before $TCM_{j'}$ if $P_{j/i} > P_{j'/i}$. Therefore, there are N_i strategies to be considered for DID_i. The rth strategy, $1 \le r \le N_i$, is the one in which the r most likely TCMs are replaced in the first call.

For the entire set of DIDs, the replacement strategy is completely specified by a k-element vector V such that V_i, $i = 1, 2, \cdots, k$, is the number of TCMs replaced in the first call for the DID_i. If the V_i most likely TCMs of DID_i are replaced in the first call, then the probability of fixing the failure for DID_i in the first call, α_i, is given by

$$\alpha_i = \sum_j P_{j/i},$$

where j belongs to the set of V_i most likely TCMs of DID_i. The probability of second call for DID_i is $1 - \alpha_i$. The average number of TCMs for the two calls for DID_i, M_i, is given by

$$M_i = \alpha_i V_i + (1 - \alpha_i) N_i.$$

Therefore, the overall average number of TCMs per failure, $\overline{M}$, and probability of second call, $\overline{P}_2$, across all k DIDs are given by

$$\overline{M} = \sum_{i=1}^{k} Q_i M_i, \text{ and } \overline{P}_2 = \sum_{i=1}^{k} Q_i (1 - \alpha_i).$$

Both $\overline{M}$ and $\overline{P}_2$ are functions of V. The replacement strategy that minimizes $\overline{M}$ subject to the condition that $\overline{P}_2$ is less than or equal to a specified very small value meets the objectives of the SFR policy. This strategy is identified by the algorithm given in [15].

The SFR policy is implemented by associating an SFR bit with each TCM called by a DID. The SFR bit is set to 1 if the TCM is to be replaced in the first call for the DID and is set to 0 if it is to be replaced in the second call. These bits are set according to the optimal SFR policy identified previously and are permanently stored on the Processor Controller File.

When the CE comes to repair a hardware-detected intermittent failure for which multiple TCMs are called, the TCMs that are to be replaced in the first SFR call are identified by diagnostics and replaced by the CE. The diagnostic microprograms keep a record of the TCMs replaced. Two situations can arise when not all TCMs are replaced in the first call: either the faulty TCM is replaced, or it is not. We define that the nonreplacement case is implied if the same error is detected within two weeks of the replacement of TCMs in the first call. If no error is detected in that period, any subsequent occurrence of an error in the DID is treated as an occurrence of a new fault, because beyond two weeks the probability of a second failure is comparable to that of missing the failing TCM in the first call.

Diagnostic (isolation) effectiveness

One of the major requirements was that the average-number-of-TCMs-called-per-detected-TCM-failure had to

be 1.2 TCMs or less. The average number of TCMs per failure is a function of the percentage of faults that are solid faults. On the basis of the technology of the 3081 and experience on previous machines, 25% of the faults are assumed to be solid faults for the 3081. A secondary objective was to isolate a high percentage of the failures to a single TCM on the sole basis of the data captured at the time of error detection. This would limit the percentage of failures where a portion of the machine would not be available to the customer while diagnostics are running and would result in improved availability. This second objective means that a high percentage of the failures should be isolated to a single TCM by a combination of direct and intersection isolation. To evaluate the fault-isolation process in terms of these objectives, we first define certain concepts that enable us to calculate the isolation effectiveness. The actual evaluation is based on statistical sampling since this is a more cost-effective method than the analytical technique.

• *Analytical definition of effectiveness*

Any given area has a known failure rate. If C_i is the failure rate of area i and the failure of this group of circuits results in N_i TCMs to be called by the fault-isolation process, then the average number of TCMs per failure, $\overline{N}$, is given by

$$\overline{N} = \frac{\Sigma N_i C_i}{\Sigma C_i}.$$

Let $D_i = 1$ if, and only if, a failure in area i can be isolated to one TCM by direct and intersection isolation, and 0 otherwise. Then the average percentage of failures isolated to one TCM by direct plus intersection isolation (P_{DI}) is given by

$$P_{DI} = \frac{100\Sigma D_i C_i}{\Sigma C_i}.$$

Thus knowing N_i, D_i, and C_i we can calculate the necessary statistics. The 3081 Processor Unit consists of blocks (groups of circuits), each of which satisfies the criteria defined for an area. But there are hundreds of thousands of these blocks and an automated computer program would be required to determine N_i, D_i, and C_i for each block. Such a program could not be justified because of the effort required to develop it. Therefore, a statistical sampling approach was used to determine diagnostic effectiveness.

• *Statistical technique*

The statistical technique is based on sampling theory [16]. A random sample of n blocks is chosen from the total set of blocks in the 3081 Processor Unit such that the probability of choosing a block is proportional to its circuit count. The isolation of a failure of a block i is determined by the following rules:

1. If the block is in a set of DIDs such that there is a single TCM common to all DIDs, then for this block $N_i = 1$ and $D_i = 1$.
2. If the block is in a set of DIDs such that there is more than one TCM in the intersection of these DIDs, it is randomly assigned to one of the two following categories, with 25% probability that a block will be in *Category 1* (stuck fault) and 75% probability that it will be in *Category 2* (intermittent fault). For each block in Category 1, N_i is the number of TCMs called by VTs. This is available from the information produced by programs that generate VTs. For each block in Category 2, N_i is given by the SFR policy for the DID. Thus N_i is assigned according to the category in which the block falls. D_i is 0 for all blocks.

The sample mean and standard deviation are calculated and the 95% confidence interval is calculated from them. The formulas for calculating these intervals are as follows:

a. The average number of TCMs per failure is in the interval given by the following relation, with 95% confidence:

$$m \pm 1.96\frac{S}{\sqrt{n}},$$

where $m = \frac{\Sigma N_i}{n}$ and $S = \sqrt{\frac{\Sigma(N_i - m)^2}{n-1}}$.

b. The probability that a given failure is isolated to one TCM by direct and intersection isolation lies in the interval given by the following relation, with 95% confidence:

$$P \pm 1.96\sqrt{\frac{P(1-P)}{n}}, \quad \text{where} \quad P = \frac{\Sigma D_i}{n}.$$

A sample size of 662 failures was analyzed. The sample average number of TCMs per failure was 1.19, and the 95% confidence interval was 1.15 to 1.23 TCMs per failure. The estimated mean percentage of failures isolated to one TCM by direct and intersection isolation indicated that the design goals were met.

After the diagnostic package was developed, it was tested by bugging the hardware. These tests indicated very good correlation between measured and projected effectiveness.

Summary and conclusion

In this paper, we have presented the approach to the diagnostics of the 3081 system implemented in a new TCM-based technology. We developed and implemented a diagnostic methodology that automatically identifies the failing TCM in a customer environment. A novel feature of this

methodology is that diagnostic microprograms isolate intermittent as well as solid failures. A high percentage of the time, the failing TCM is identified without running any tests on the hardware. It was also shown how we met the *a priori* stated objectives for the diagnostic package.

The main trend in integrated circuit technology is to higher-and-higher-density packages (such as TCMs) to give users highly reliable and available machines that are easy to repair. An integrated approach such as the one presented in this paper is mandatory for meeting those objectives and for keeping service costs at a reasonable level.

Acknowledgments

The authors wish to acknowledge the work of J. H. Miller on defining a procedure for obtaining direct-isolation domains. Thanks are due to D. W. Krueger for providing an example of a direct-isolation domain, and to H. Schiller and H. C. Long for reviewing the paper and providing helpful comments. The members of the Diagnostic Development group implemented the strategy and participated in evaluating the isolation effectiveness.

References

1. E. B. Eichelberger and T. W. Williams, "A Logic Design Structure for LSI Testability," *Proceedings of the 14th Design Automation Conference*, New Orleans, LA, June 1977, pp. 462–468.
2. J. Pomeranz, R. Nijhuis, and C. Vicary, "Customized Metal Layers Vary Standard Gate-Array Chip," *Electronics*, 105–107 (March 15, 1979).
3. H. W. Curtis, "Integrated Circuit Design, Production, and Packaging for System/38," *IBM System/38 Technical Developments*, 11–15 (1978), Order No. G580-0237, available through IBM branch offices.
4. A. J. Blodgett and D. R. Barbour, "Thermal Conduction Module: A High-Performance Multilayer Ceramic Package," *IBM J. Res. Develop.* **26**, 30–36 (1982, this issue).
5. *A Guide to the IBM 3033 Processor Complex, Attached Processor Complex, and Multiprocessor Complex of System/370*, Order No. GC20-1859, available through IBM branch offices.
6. R. S. Swarz, "Reliability and Maintainability Enhancements for the VAX-11/780," *Proceedings of the 1978 International Symposium on Fault-Tolerant Computing, FTCS-8*, Toulouse, France, June 21–23, 1978, pp. 24–28.
7. A. Shah, "A Technique for Failure Isolation in a Large Computer System," *Proceedings, IEEE ICCC80*, Rye, NY, October 1980, pp. 238–240.
8. D. C. Bossen and M. Y. Hsiao, "Model for Transient and Permanent Error-Detection and Fault-Isolation Coverage," *IBM J. Res. Develop.* **26**, 67–77 (1982, this issue).
9. F. F. Sellers, Jr., M. Y. Hsiao, and L. W. Bearnson, *Error Detecting Logic for Digital Computers*, McGraw-Hill Book Co., Inc., New York, 1968.
10. John Reilly, Arthur Sutton, Robert Nasser, and Robert Griscom, "Processor Controller for the IBM 3081," *IBM J. Res. Develop.* **26**, 22–29 (1982, this issue).
11. W. C. Carter, H. C. Montgomery, R. J. Preiss, and H. J. Reinheimer, "Design of Serviceability Features for the IBM System/360," *IBM J. Res. Develop.* **8**, 115–126 (1964).
12. P. S. Bottoroff, R. E. France, N. H. Garges, and E. J. Orosz, "Test Generation for Large Logic Networks," *Proceedings of the 14th Design Automation Conference*, New Orleans, LA, 1977, pp. 479–481.
13. T. W. Williams and K. P. Parker, "Testing Logic Networks and Designing for Testability," *IEEE Computer* **12**, No. 10, 9–22 (October 1979).
14. N. C. Berglund, "Processor Development in the LSI Environment," *IBM System/38 Technical Developments*, 7–10 (1978), Order No. G580-0237, available through IBM branch offices.
15. R. Rutledge and J. A. Santillo, IBM Corporation, Poughkeepsie, NY, private communication (1977).
16. W. A. Cochran, *Sampling Techniques*, 3rd edition, John Wiley & Sons, Inc., New York, 1977, Chs. 2 and 3.

Received December 14, 1979; revised July 9, 1981

The authors are located at the IBM Data Systems Division laboratory, Poughkeepsie, New York 12602.

RECOVERY FROM MACHINE ERRORS

VIA

INSTRUCTION RETRY

AND

FUNCTIONAL RECOVERY

Prepared by

JUPITER MAINTAINABILITY ENGINEERING GROUP

Gary Clearwater
Rick Ellison
Jutta Josbaecher
Stu Kennedy
Bob Maloney
Dave Rockwell
Nigel Webb
Don Werth

1.0	SCOPE	4
2.0	MACHINE ERRORS	4
2.1	DEFINITIONS AND EFFECTS	4
2.2	CAUSES	5
2.2.1	SEMICONDUCTOR FAILURES	5
2.2.2	PRODUCTION PROCESS DEFECTS	5
2.2.3	FIELD OPERATION AND USAGE	6
2.3	FREQUENCY OF OCCURRANCE	7
2.4	PREDICTION TECHNIQUE	8
3.0	SOLUTIONS FOR MACHINE ERRORS	9
3.1	TEST PROCESSES AND SCREENING	10
3.2	FAULT TOLERANCE	10
3.2.1	FUNCTIONAL RECOVERY	11
3.2.2	INSTRUCTION RETRY	11
3.2.3	REDUNDANCE/RECONFIGURATION	11
4.0	CUSTOMER DEPENDENCY	12
4.1	CUSTOMER SATISFACTION	12
4.2	THE METRICS TODAY	15
4.3	SUPPORTING ELEMENTS	15
4.4	OPERATING SYSTEM	16
5.0	HARDWARE ERRORS AND THEIR MANIFESTATIONS	16
6.0	INDUSTRY TRENDS TO SUPPORT FAULT TOLERANT DESIGN	18
6.1	IBM SYSTEMS	18
6.2	UNIVAC	18
6.3	HONEYWELL DPS-8 And DPS 88/81	19
6.4	OTHER SYSTEMS	19
7.0	SUMMARY AND CONCLUSIONS	20

APPENDIX A REFERENCES

1.Ø SCOPE

We will identify the causes of machine errors, their frequency, and their effect on system reliability and customer satisfaction. We will also discuss various techniques, and their inherent value in overcoming the most prevalent types of machine errors.

2.Ø MACHINE ERRORS

Errors are produced by faults that exist in the machine for various reasons. A fault is defined as "an erroneous state of hardware or software resulting from failures of components, physical interference from the environment, operator error, or incorrect design".[SIEWIOREK and SWARZ, 1982]

There are three main types of faults to be considered, that produce machine errors.

2.1 DEFINITIONS AND EFFECTS

- SOLID - describes a type of fault which is continuous and stable. Generally they are well suited to the level of fault isolating diagnostics that have been engineered in previous products such as the KL1Ø. This type of fault is not often recoverable (without redundancy and dynamic resource switching) and as such, almost always results in a system crash.

- INTERMITTENT - describes a fault that is only occasionally present due to unstable hardware or varying hardware or software states (for example, as a function of load or activity). Faults of this nature are potentially detectable and repairable by parts replacement, given that the design allows for it. Prediction of these faults is very difficult and no standard exists today (part of the reason is that no large effort has been mounted to design in circuits to easily monitor for these faults), although some attempts have been made to record them in special test cases. [O'NELLIE, E. J., and J. R. HALVERSON]

- TRANSIENT - describes a fault resulting from temporary environmental conditions such AC/DC power fluctuations, RF noise, temperature/humidity changes, or hardware/software design errors. These faults, much like the intermittent, have one very important difference, they cannot be repaired by parts replacement because the hardware is left

physically undamaged.

2.2 CAUSES

The reasons for failures in electronic parts and assemblies start in the fabrication and build process, and continue through the entire life of the machine. The basic reliability of a design degrades as the transition from paper design to production to field operations occurs. The three sections that follow, illustrate some of the major reasons.

2.2.1 SEMICONDUCTOR FAILURES -

There are numerous ways in which a semiconductor can fail resulting from defects in the manufacturing process, and also stress during normal operation. The following chart summarizes die-related defects:

Classification	Relative Percent
SURFACE	16
BULK	7
OXIDE	14
DIFFUSION	8
METALIZATION	51
I/O CIRCUIT	4

[SIEWIOREK and SWARZ, 1982, PG 25]

2.2.2 PRODUCTION PROCESS DEFECTS -

As an example of some types of defects, consider a resistor with its leads bent close to its body. If the stress imposed during bending caused the body to chip, this is a quality defect. However, had the stress been inadequate to chip the body, the defect would go unnoticed by conventional inspection. When the body is cycled through a temperature range, small cracks can develope in the body. This would allow moisture and other gases to contaminate the resistive element causing resistance changes. This is a reliability

defect. This defect is also a design defect if the design specifications require a tight bend to fit the component properly on the board. If the improper bend is due to poor workmanship, the defect is classified as an induced defect. Some other examples are:

PROCESS	INDUCED DEFECTS	FAILURE MODE(S)
Wire Stripping	Nicked lead	Open Circuit
	Short lead	Intermittent/Open/Noise
Soldering	Excessive Heat	Open/Value Change/Noise
	Insufficient Solder	Intermittent/Open/Noise
Wire Wrapping	Broken Wire	Open
	Loose Connection	Intermittent/Open/Noise

NOTE

For more information refer to the "RELIABILITY DESIGN HANDBOOK" March 1976 - table 4-43, pg 253 - Rome Air Developement Center

2.2.3 FIELD OPERATION AND USAGE -

Degradation in reliability also occurs as a result of system operation and maintenance. Operators in the field will stress systems beyond the predicted levels either through neglect, unfamiliarity with the equipment, carelessness, or operational constraints. For example, while we do not expect willful neglect on a system of our complexity and cost, that very same complexity and cost dictates that the customer use the system very effectively. Many jobs become critical with respect to deadlines and to meet them he will sometimes operate the system in a temporarily degraded environment. (this is where the neglect appears) In some cases the environment may remain degraded for extended periods of time (eg. waiting for the air conditioner part required).

Damage to the system that results from a poor environment may not be easily perceived immediately after occurance. Subtle changes may have occurred which serve to degrade system and circuit functions at a later time. When transients also occur, the combination of both effects can produce faults/errors that would not have otherwise been evident.

Improper maintenance can also induce defects. During unscheduled maintenance (system down) spare modules may be used to locate the defective module, with the assumption that the spares complement is functionally without fault/error. This is not always the case, and a spare module with a different fault may be left in the system, thereby compounding the problem. Other defects result from:

- Foreign objects left in an assembly
- Dirt injection

2.3 FREQUENCY OF OCCURRANCE

TRANSIENT and INTERMITTENT faults are already a major source of errors in systems. An early study for the U.S. Air Force [Roth et al., 1967a] showed that 80% of the failures in electronic computers are intermittent. Another study by IBM [Ball and Hardie, 1967] indicated that intermittents comprised over 90% of field failures.

A more recent study was conducted [O'Nellie,E.J., and J.R. Halverson, June 1980] by Sperry Univac Defense Systems Division for NASA. One particular computer was selected (composed of 8098 SSI integrated circuits) which had the most equipment utilization reports as well as malfunction data. These were examined to understand the relationship between intermittent and hard failures. The conclusions are:

- A strong positive correlation exists between the number of chips on a board and the number of intermittent failures for that board.
- The data in the report suggests that the hard failure rate decreased with time.
- The rate of intermittent failures appeared to increase with time.
- A conservative estimate is that transient/intermittent failures occur 10-20 times more frequently than hard failures.

- One third of all hard failures start out as intermittent failures.

2.4 PREDICTION TECHNIQUE

The estimation of predicted behavior for solid faults, given the number and type of gates in a design, as well as their expected environment, among other things, is accomplished with a model known as MIL-HDBK-217. This model has been developed as a result of extensive studies on electronic component failures funded by the U.S. Department of Defense, Reliability Analysis Center, Griffiss Air Force Base, N.Y. It is considered by the industry to be the "bible" for predicting component failures.

The KL10 was modeled using revision "B" of the model. Subsequent measurement of in-field failures (using the Labor Activity Reporting System - LARS) proved to very close to the prediction.

KL10 MTBF

PREDICTED	MEASURED
.9 months (657 hrs)	.9-1.2 months (876 hrs)

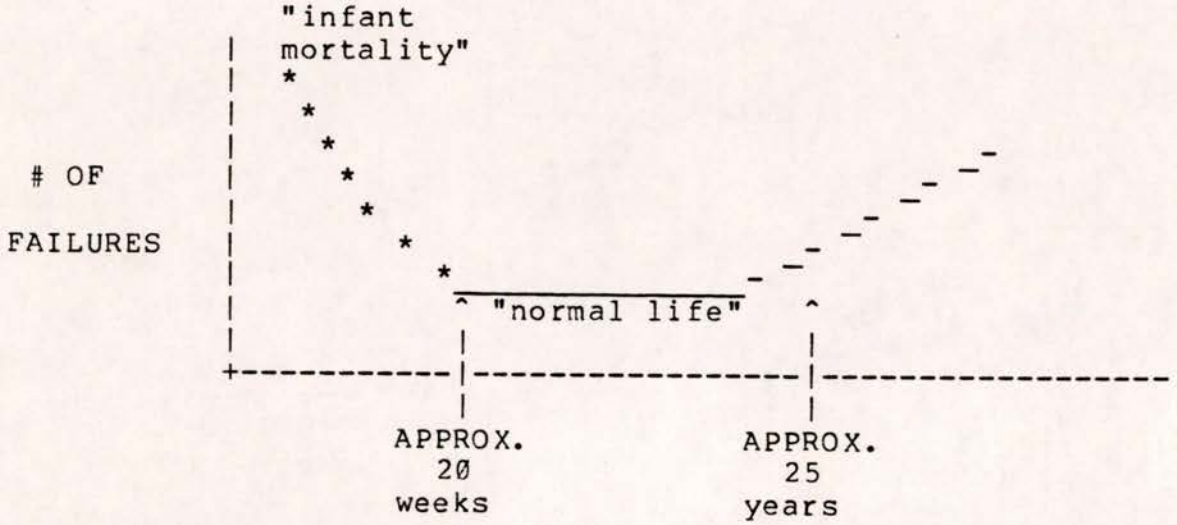
Actual measured performance was somewhat better than the predicted performance. However, errors and inconsistencies were discovered in the model. This has led to refinements which now have bumped the model to revision "D". The following example will illustrate the results of these refinements:

KC10 MTBF PREDICTION

MIL-HDBK- 217C	MIL-HDBK-217D
800 hrs	1196 hrs

3.0 SOLUTIONS FOR MACHINE ERRORS

The life of an integrated circuit is divided into three stages; infant mortality period, normal lifetime, and wear-out period. These three stages are represented in a life characteristic curve (sometimes called a bathtub curve) [Siewiorek and Swarz, 1982, pp 8-9] as follows:



Notes -
- The number of failures is very high in early life
- The failure rate is quite stable during useful life
- This is a summary of all chip technologies

Our own field experience also tends to support the life cycle curve, based on a study of service calls made during the warranty period following installation and the subsequent 90 day period. It shows that there are 60% more calls during the first period than the second, which suggests a similiar curve.

There are a number of techniques/processes/tests that can be, and are, used to overcome these failures. The actual one used depends on the time at which the technique is applied.

3.1 TEST PROCESSES AND SCREENING

During the early life of a component, vendors perform various screening type tests designed to detect manufacturing defects:

- Internal visual
- Stabilization bake
- Thermal shock
- Temperature cycle
- Burn-in

NOTE

These are only a few of all possible tests - others and their most effective applications can be found in the "RELIABILITY DESIGN HANDBOOK" tables 4-22, and 4-23

However, not all the defects will be found in this initial testing period. Even for the most stringent reliability (Class A), the burn-in time is only 240 hours, and not all the failures will fall out. Those that remain will be found by the system manufacturer either in manufacturing testing or eventually in normal field operation.

3.2 FAULT TOLERANCE

The rate of failures remains relatively constant during the normal life of the system in its customer environment and application. At this point further testing by Field service is prohibited by the amount of time that would be required, since the customer expects to use his system all the time. Therefore another method must be used to mask or circumvent the failures that remain in the system, generically called "fault tolerance".

Remember that the great majority of faults/errors will be of the intermittent/transient variety, so the solution that makes sense is to attempt to recover the failures that are of short duration.

This can be accomplished in two general ways.

3.2.1 FUNCTIONAL RECOVERY -

The choice of using this method would depend basically on how effective and complex the solution will be. Some machine functions lend themselves very well to this type of recovery, for example, ECC on memories. Others such as recovery of a bus cycle that experienced a parity error, may be difficult if all or a portion of the machine must be temporarily halted and then re-started. In this case, it may be easier to use another method.

3.2.2 INSTRUCTION RETRY -

Rather than attempt to get the machine into the correct attitude to perform a very granular functional recovery for all cases, It may be easier to re-execute the instruction, starting back at the beginning of the instruction when possible. Thus, the design could be simplified because all of the timing and other details of the exact nature of the error need not be remembered and re-created to attempt recovery.

Further gains can be made with this recovery method when, if the instruction cannot be successfully executed, it is known what process/program called it. The best case would be an individual user's program, where only that user would be terminated, while keeping the other users alive and well.

3.2.3 REDUNDANCE/RECONFIGURATION -

An additional solution that can be combined with either of the above is to provide redundant circuits and functions. This is useful for further extending the perceived reliability of the system in that it is possible to tolerate frequently re-ocurring faults or solid faults by switching to functionally identical backup circuits. Examples would be AC sets, and CACHE directory.

Memories, Caches, Translation buffers and other arrays can be designed in sections so that a bad section can be turned off when it begins to produce higher than usual error rates. This not only reduces overhead in error recovery time, but also allows for graceful degradation rather than abrupt system crashes. The customer could make a decision, based on his application and availability of Field Service assistance, when to stop processing and effect a deferred repair.

4.0 CUSTOMER DEPENDENCY

In the past customer dependence on computer resources mainly consumed a ten hour prime-time shift. The remaining hours were scheduled for system maintenance. Extensive amounts of down time were often expended to resolve intricate hardware/software problems. During this period a fault would resurface several times before it could be identified to a field replaceable unit or a software patch installed.

Today the dependency on computer resources has escalated. LSG customers are running three operating shifts to utilize the large investment in their system. This scenario has reduced scheduled down time for corrective and preventive maintenance, necessitating the on-line recovery of system faults.

The systems of the mid 80's thru early 90's will require minimal down time for maintenance. The system's ability to recover from error will no longer be just marketing leverage used as an advantage over the competition, error recovery will be mandatory. With increased system thru-put the number of interactive users will also substantially rise, hence each system crash will affect larger and larger numbers of users and processes. This problem becomes compounded with the implementation of loosely coupled systems where multiple computers now depend on other computers to be up.

4.1 CUSTOMER SATISFACTION

Customer Satisfaction with our products is a major goal of what we want to achieve with CSSE's efforts. We have in the past loosely equated System Availability with Customer Satisfaction. Availability has been found to be a relatively poor indicator of Customer Satisfaction because even though a customer may have a 98% Availability, he may be very dissatisfied. The cause is usually found to be that the system is crashing a lot, coming right back up, running for a while, crashing again, coming up, running a while, and so on. Since Availability is, loosly,

$$\text{Availability} = \frac{\text{Scheduled Uptime - downtime}}{\text{Scheduled Uptime}},$$

one can see that with 5 minutes down time per crash (reload time) and a scheduled uptime of 168 hours per week (24x7) which many customers require, the system can crash 20 times per week and 99% availability will still be met. Yet the customer will NOT be happy!

If we assumed a uniform distribution of the 20 crashes, we would see one every 8.4 hours.

Studies of reliability for the military have come up with metrics which can be applied to the computer business.

Availability can be defined as the probability that the system will be up when you want TO START your job.

Reliability can be defined as the probability the the system will stay up long enough to finish your job.

System Effectiveness can be defined as the probability that the system will be BOTH Available AND Reliable enough to start your job when you want AND to finish it without interruption.

Some equations have been derived to allow us to crank numbers through these definitions. (I have substituted what I belive to be analogous parameters for the actual parameters from the formal definitions).

$$\text{Avail.} = \frac{\text{Mean Time To Crash (i.e., MTTC)}}{\text{MTTC + Mean Time To Recover and Repair (i.e., MTTRR)}}$$

note: MTTC + MTTRR = 100% of time customer expects to have his system.

$$\text{Reliab.}(t) = e^{\left(\frac{-t}{\text{MTTC}}\right)}$$

note: t is the time required to complete the task being measured.

t-hour System Effectiveness = Avail. * Reliab.(t)

For example, an 8-hour System Effectiveness of 99% means that there is a 99% probability that the user will be able to sit down, start, and finish an 8-hour job without an interruption. Or for dealing with system managers - the probability that the system will be up and stay up for just one shift.

The following table shows various values of the parameters

Note that MTTRR = Mean Time to Recover and Repair (when repair is needed)

System Effectiveness	Availa-bility	Relia-bility	MTTC Weeks	MTTC Hours	MTTRR Hours	
8 hour 90.81%	99.88%	90.92%	0.50	84	0.1	Twice a week
8 hour 92.22%	99.90%	92.31%	0.60	100	0.1	JUPITER MTTC
8 hour 95.29%	99.94%	95.35%	1.00	168	0.1	Once a week
8 hour 98.39%	99.98%	98.41%	2.98	500	0.1	
8 hour 97.63%	99.21%	98.41%	2.98	500	4.0	
8 hour 98.41%	99.47%	98.94%	4.46	750	4.0	
8 hour 99.00%	99.67%	99.33%	7.12	1196	4.0	
8 hour 99.05%	99.68%	99.36%	7.44	1250	4.0	

note: The .1 hour MTTRR indicates the expectation that for short MTTC's the problem is NOT a solid fault and that a simple monitor reload affords sufficient time for the hardware to re-stabilize. The 4 hour MTTRR indicates that for longer MTTC's the reason for a crash is most likely a solid fault which requires a Field Service on site visit.

Some interesting obeservations can be made:

If we are crashing twice a week (MTTC = 84 hrs.) and the system is down only 6 minutes to reload the monitor each time, we achive 99.88% Availability, but only 90.81% System Effectiveness.

If we crash once per week (slightly better than we do on KL10's today), we achieve 95.29% System Effectiveness.

We CANNOT achieve a 99% 8-hour System Effectiveness until our Mean Time To Crash reaches almost 1200 hours.

4.2 THE METRICS TODAY

As mentioned previously, the most meaningful criteria that we measure, in terms of customer satisfaction, is the number of system crashes that customers experience. The "CRASH and UPTIME REPORT" is completed on a weekly basis by the service representative, approved by the customer, and then forwarded to us. We compile and archive them and produce a "roll-up" report which gives a global picture. The KL10 system crash rate (MTTC), as measured today is 131 hours.

System crashes are also divided into four areas; hardware, software, environment and others (unknown causes). Listed are the ratios of each element.

57%	hardware	- solid fault - transient fault - intermittent fault
18%	software	- bug
14%	environment	- power - cooling - radiation - mechanical
11%	other	- operations - application

These show current distribution of problems reported by the field. It is easy to see that the biggest place to make improvement is in the hardware and this is where instruction retry and other recovery mechanisms can most significantly help.

4.3 SUPPORTING ELEMENTS

The environment of a computer system can contribute to as much as 14% of system failures. To the extent that you can control the environment, you can influence the occurrence of such failures. Factors such as power, temperature, grounding and humidity control must be considered during the early stages of system design and also during site preparation. Field service verifies that the site must conform with the environmental specification before an uptime guarantee service contract is approved. However, it also important to anticipate environmental changes that occur, and design to be able to tolerate them rather than crash the system.

4.4 OPERATING SYSTEM

Software integrity is the ability of the software to do what it is designed to do and generate valid data. Software integrity differs from hardware reliability in that it is mainly a function of experience or maturity. That is, the more experience the software has had, the less likely it is to have any unforeseen problems. Software integrity can be evaluated in terms of the quanity and quality of vendor testing, and the diversity and complexity of the application using the software.

5.Ø HARDWARE ERRORS AND THEIR MANIFESTATIONS

The hardware relibility of a system is affected by numerous factors. Models are available to calculate the solid failure rate, however a majority of hardware failure modes are at least initially intermittent,and some may never become solid. Industry data varies, but in general ratios of 2Ø to 1 for intermittent versus solid phenomena have been mentioned previously.(refer to 2.3)

It is not possible to accurately measure the intermittent failure rate for the KL1Ø because the design does not allow it (ie. you must be able to first detect it, then recover from it, and then finally count it). We could back into an estimate by making an inference between the ratio of system crashes to solid errors (both of which have been measured) which is roughly 7 to 1.

MTTC(hrs)	MTBF(hrs)
131	876

NOTE

This is not however, the same as the intermittent/solid ratio and should not be mistaken as such. It is only a starting point.

What we expect to see for the KC1Ø is more in line with industry experience that shows, as systems become more miniaturized and complex, they also exhibit more intermittent tendencies. Some reasons for this are:

- Shrinking device dimensions
- Lower energy levels for indicating logic levels

o Higher speed operation

Approximately four years ago, IBM announced a major committment to symptom directed diagnosis, specifically to overcome intermittent faults(recovery), as well as provide other advancements in testability and design. [John Shebell, "Customer Services Systems Engineering, Strategic Plan, 1983", Chapter 4, Technology Appraisal, pg 45]

The following example attempts to show what we believe the effect of intermittents will be:

EXAMPLE

The MTBF calculated for the KC10 as designed, is 1196 hours. In other words, a crash should be expected to occur at that frequency. However, intermittents and transients will occur, that also cause system crashes, and therefore must be considered.

Assume...(conservatively)

that, 75% of all intermittents cause a crash...

that the ratio of intermittent/solid faults is 15/1

Then...

1196/15 = 80 hours per error occurrance

And... 80 (hrs per hit) x 1.25 (multiplier for 75% crash hit rate)

Equals... MTTC (mean time to crash) of 100 hours

Implementing instruction retry would reduce the number of unscheduled system interruptions. The predominant number of faults are intermittent in nature rather than being solid; therefore there is a high probability of success of execution and recovery if an instruction retry can be attempted.

One justification of this position can be found by way of the study of the IBM 370/168 [Iver and Rossetti, 1982] that concludes "that in only 14% of the errors does the operating system stop processing". That is, in 86% of the cases, the system successfully recovered from errors.

6.0 INDUSTRY TRENDS TO SUPPORT FAULT TOLERANT DESIGN

6.1 IBM SYSTEMS

All systems currently produced by IBM have some form of instruction retry. The 4300 series and the new 308X series have extensive instruction retry logic. The 303X series and the larger 370 systems have instruction retry capability.

The 4300 series which uses the level sensitive scan design (LSSD) continuously saves and periodically purges data that is needed for instruction re-execution. Instruction are retried by restoring information from a retry array.

If retry is successful the processor unit will generate a machine check interrupt code (MCIC) and continue normal operation. If retry is not successful the MCIC will contain information to aid the operating system in it`s own error handling procedure.

The new IBM 3081 series systems have an even more extensive instruction retry capability. It too uses the LSSD technique to capture error information. 35,000 points in the machine can be read out using LSSD.

One difference from the 4300 series is the 3081 uses a hardware checkpoint mechanism. The machine will establish a checkpoint at some instruction and will save the contents of the architected registers (PSW,GR`s,etc.) in backup registers. Any store into the cache will cause the old value of the changed word to be saved in a pushdown array. In event of an error condition the architected registers will be restored to their original value and the cache will be returned to its original condition by storing back the contents of the pushdown array in the reverse order.

Regular checkpoints are established during normal instruction execution under specific machine conditions with no effect on CPU performance. By use of this technique up to 20 instructions may be restored to original conditions and retried.

A recent study of an IBM 3081 at SLAC (Stanford Linear Accelerator Center) indicates the system is able to successfully recover from 80% of the errors it detects.

6.2 UNIVAC

The UNIVAC 1100/60 system uses instruction retry in transient/intermittent error recovery. When a fault is detected micro-execution is halted and a fault recovery micro-routine is entered. This will analyze the failure and build a fault status word. It will restore the general registers and machine state from a "save storage memory element" which contains the original operands for the instruction.

The majority of instructions can be retried using this procedure. If retry is unsuccessful the service processor is interrupted and will take further corrective action ,including "program transplant" where the failing instruction processor is removed from the system configuration and the program is switched the the other instruction processor.

6.3 HONEYWELL DPS-8 And DPS 88/81

These systems both use instruction retry in their fault tolerant design. They use a "history register" which contains the critical data for the last 64 micro cycles. When an error occurs an interrupt to the operating system occurs and a program called "HEALS" is executed. This will restore the data from the history registers and attempt re-excution of the instruction in error.

6.4 OTHER SYSTEMS

Both the AMDAHL 470 and 580 series systems use instruction retry for fault recovery. Machine check handling analyzes errors and attempts recovery by retrying the failed instruction if possible. Both these systems are very large pipeline design machines.

The large NATIONAL ADVANCED SYSTEMS AS/7000 and AS/9000 both have instruction retry capability. The NAS systems are the most popular of the IBM plug compatible mainframes and are manufactured by Hitachi. They are rated by users as the most reliable of all large systems.

The NIPPON ELECTRIC CO. (NEC) ACOS system 800/900 has extensive instruction retry logic in the execution processing unit. This is a 200,000 gate CPU that is equivalent to an IBM 370 168 or 3033. Because of its fault tolerant design and the instruction retry capability it is able to recover from 90% of EPU failures. [Katsuichi Tomita, Shungi Tashiro, and Takuo Kitamura, 1980]

NEC has also just announced the ACOS 1000 system which is designed to compete against the largest Honeywell DPS 88 systems. It is certain that this system has even more fault tolerance design then the 800/900 series.

7.0 SUMMARY AND CONCLUSIONS

1. KC10 best case predicted MTBF (i.e., solid fault) is 1196 hours (7.1 weeks).

2. Intermittent to Solid fault ratio is 15:1, and 75% of all intermittents cause crashes. Therefore, expect intermittant fault once every 100 hours (4.2 days).

3. Without Instruction Retry, we will crash at the intermittent rate. Consequently, the System Effectiveness data would look like:

System Effectiveness	Availability	Reliability	MTTC Weeks	MTTC Hours	MTTRR Hours
8 hour 92.22%	99.90%	92.31%	0.60	100	0.1

note: 6 minute reload time because system crashed; not solid fault.

4. With Instruction Retry, we will crash at the MTBF (i.e., Solid Fault Rate) of 1196 hours (7.1 weeks), and the System Effectiveness data would look like:

System Effectiveness	Availability	Reliability	MTTC Weeks	MTTC Hours	MTTRR Hours
8 hour 99.00%	99.67%	99.33%	7.12	1196	4.0

note: 4 hour repair time because solid fault.

5. Software Crashes are not included here, but they only make matters worse because customers are reaching the point where they don't care whether the system crashed for hardware or software reasons. DEC is also moving toward "System Service" contracts which cover both hardware and software.

6. All of our competition have instruction retry.

In conclusion, it is only a short matter of time until our customers realize that 99% availability isn't what they need -- 99% effective is what they need. Our product had better be able to meet those needs, or our competition will get the business.

APPENDIX A

REFERENCES

[Siewiorek and Swarz, 1982] Siewiorek, D. and R. Swarz, "The Theory and Practice of Reliable System Design", Digital Press, 1982

[O'Nellie, E. J., and J.R. Halverson], "Study of Intermittent Field Hardware Failure Data In Digital Electronics", NASA Contract Report 159268, Langely Research Center, Hampton, VA. 23865, June 1980

[R. T. Anderson], "Reliability Design Handbook", March 1976, Catalog No. RDH-376, Under Contract to: Rome Air Developement Center, Griffiss Air Force Base, NY 13441

[Roth et al., Roth J. P.; W.G. Bouricius; W.C. Carter; and P.R. Schneider.] "Phase II of an Architectural Study for a Self-Repairing Computer". SAMSO-TR-67-106. U.S. Air Force Space and Missile Division, El Segundo. CA.

[Ball, M. O. and F. Hardie] "Effects and Detection of Intermittent Failures in Digital Systems". IBM 67-825-2137. 1967.

[Iyer and Rossetti, 1982] Iyer, R. K. and D. J. Rossetti, "A Statistical Load Dependency Model for CPU Errors at SLAC", Digest 12th International Symposium on Fault-Tolerant Computing, June 1982, pp. 363-372.

[John Shebell, "Customer Services Systems Engineering, Strategic Plan, 1983", Chapter 4, Technology Appraisal]

[Katsuichi Tomita, Shungi Tashiro, and Takuo Kitamura, 1980] "A Highly Reliable Computer System - Its Implementation and Result", Digest 10th International Symposium on Fault-Tolerant Computing, 1980.

[Mark L. Ciacelli 1981] "Fault handling on the IBM 4341 processer" Digest of papers 11th International Symposium on Fault-Tolerant Computing, 1981.

[L.A. Boone, H.L. Liebergot, R.M. Sedmark 1980] "Availability, Reliability, and Maintainability Aspects of the Sperry Univac 1100/60" Digest of papers 10th International Symposium on

Fault-Tolerant Computing, 1980. pp. 3-8

[M.S. Pittler et al. 1982] "IBM 3081 System Development Aspects" IBM Journal of Research and Development Vol. 26 No.1 January 1982

[P.G. Depledge 1981] "Fault-Tolerant Computer Systems" IEEE proceedings Vol.128 No.4 May 1981.

[Rosetti and Iyer 1982] "Software Related Failures on the IBM 3081: A Relationship with System Utilization" Technical Report No. 82-8, Center for Reliable Computing, Stanford, Calif. July 1982.