December 1988

UILU-ENG-88-2266
CSG-96

COORDINATED SCIENCE LABORATORY
College of Engineering

TESTING MEMORY MANAGEMENT UNITS

Jeffrey Hamilton
Ramachandra P. Kunda
Bharat Deep Rathi

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION: Unclassified	1b. RESTRICTIVE MARKINGS: None
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): UILU-ENG-88-2266 CSG-96	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: Coordinated Science Lab University of Illinois — 6b. OFFICE SYMBOL (If applicable): N/A	7a. NAME OF MONITORING ORGANIZATION: IBM/DARPA/SRC
6c. ADDRESS (City, State, and ZIP Code): 1101 W. Springfield Ave. Urbana, IL 61801	7b. ADDRESS (City, State, and ZIP Code): IBM Research T. J. Watson Research Center Yorktown Heights, NY 10598; DARPA 1400 Wilson Blv. Arlington, VA 22209; SRC P.O. Box 12053 RTP, NC 27709
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: DARPA/IBM/SRC — 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: IBM 12490066, DARPA N00039-87-C-0122, and SRC 87-DP-109
8c. ADDRESS (City, State, and ZIP Code): see 7b.	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)

Testing Memory Management Units

12. PERSONAL AUTHOR(S)

Hamilton, Jeff, Kunda, Ramachandra P., and Rathi, Bharat Deep

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM ______ TO ______	10-13-88	16

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Test, diagnostics, test generation, system-level test, functional test.

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This paper describes a functional diagnostics method to test the Memory Management Unit (MMU) of the RP3 system. The Research Parallel Processor Prototype (RP3) is a highly parallel computer being built at the Watson Research Center The RP3 is designed to have 512 processors, but the prototype being built has 64 processors. This system supports both private and shared address spaces. Virtual memory management is done across both of these spaces. Translation look-aside buffers are used for this vitual-to-real address mapping.

The MMU is one of the more complex modules of the RP3 processor. It provides the control for

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION: Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code) — 22c. OFFICE SYMBOL

many of the processor subsystem functions. The MMU is a multi-chip level-sensitive scan-design (LSSD) that does not support special testing hardware. Scan-based test methods could not be used for this design, because LSSD rules were only followed at the chip level. The functional diagnostics approach suggested in this paper uses the system's instruction set to diagnose the MMU. This method uses a "divide-and-conquer" approach to reduce the testing complexity. An added advantage of this approach is that the same diagnostics have been used for MMU debug, system integration and system qualification. During this process we have detected design errors, permanent faults, and even the known timing problems in the logic.

Testing Memory Management Units

Jeffrey Hamilton
IBM Research Division
T. J. Watson Research Center
Yorktown Heights, N.Y. 10598

Ramachandra P. Kunda
Computer Systems Group
Coordinated Science Laboratory
University of Illinois
1101 W. Springfield Ave
Urbana, Il 61801

Bharat Deep Rathi
IBM Research Division
T. J. Watson Research Center
Yorktown Heights, N.Y. 10598

Acknowledgement: This research is sponsored in part by the Semiconductor Research Corporation under Contract 87-DP-109, in part by Defence Advanced Research Projects Agency under contract N00039-87-C-0122, and in part by the IBM Corporation under Contract 12490066.

ABSTRACT

This paper describes a functional diagnostics method to test the Memory Management Unit (MMU) of the RP3 system. The Research Parallel Processor Prototype (RP3) is a highly parallel computer being built at the Watson Research Center The RP3 is designed to have 512 processors, but the prototype being built has 64 processors. This system supports both private and shared address spaces. Virtual memory management is done across both of these spaces. Translation look-aside buffers are used for this vitual-to-real address mapping.

The MMU is one of the more complex modules of the RP3 processor. It provides the control for many of the processor subsystem functions. The MMU is a multi-chip level-sensitive scan-design (LSSD) that does not support special testing hardware. Scan-based test methods could not be used for this design, because LSSD rules were only followed at the chip level. The functional diagnostics approach suggested in this paper uses the system's instruction set to diagnose the MMU. This method uses a "divide-and-conquer" approach to reduce the testing complexity. An added advantage of this approach is that the same diagnostics have been used for MMU debug, system integration and system qualification. During this process we have detected design errors, permanent faults, and even the known timing problems in the logic.

1.0 Introduction

In order to successfully build any system, a methodology to test the system must be defined. Several test approaches may be needed to diagnose the system at all levels of integration. Often special hardware is incorporated in the design to enable it to be easily tested. Scan-based test methods have often been used to diagnose the lower levels of integration. Although these methods provide good fault coverage, they can take a long time to execute.

In large systems like the Research Parallel Processor Prototype (RP3) [PFIS85], which consist of many subsystems, reducing the time complexity of the test method is very useful. This reduction helps qualify the large number of components used and is also helpful during system integration and qualification. The Functional Diagnostic Method (FDM) used to test RP3 [RATH88] has been very effective in both fault coverage and testing time. This paper describes how this method was used to test RP3's MMU.

FDM uses the system's instruction set to test the system. This method partitions the systems into several functions and then defines tests for each of these functions. The MMU is one of these functions. This "divide-and-conquer" approach reduces the complexity of the tests. In order to test the MMU, it is partitioned like the whole system and tests for these sub-functions were developed. Using this approach, we were able to develop the MMU diagnostics in 4 man-months. Using similar diagnostics for other portions of the system, we were able to qualify the 64 processors of the RP3 prototype in about 3 months.

This paper is organized as follows: the next section gives an overview of the RP3 system and discusses our motivation to use the FDM approach to test the MMU. Section-3 gives an overview of the MMU organization, while section-4 describes the TLB array topology. In section-5, an overview of the MMU diagnostics method is given. This is followed by a description of the fault model and test generation. Finally, we relate some of the experiences we had with these diagnostics and give a summary of our findings.

2.0 Background and Motivation

The RP3 is a highly parallel computer system being built at the IBM T. J. Watson Research Center. It has been architected to have 512 processors, 2 GBytes of memory, 192 MBytes/second I/O, and an interconnection network with a peak bandwidth of 13 GBytes/second. A prototype of the RP3 is currently being built with 64 processors. The organization of a 64-processor RP3 system is shown in Figure 1 on page 2. It can support both shared memory and distributed memory-based computing models. The main reason behind building this machine is to conduct research in the highly parallel processing area. A more detailed description of this system is given in [PFIS85]

This system uses the ROMP microprocessor [ROMP86], an IBM RISC processor, as its main computational units. In order to support the various functions defined by the RP3 architecture [PFIS85], the support chips for the ROMP processor could not be used. Therefore, the project designed its own memory management unit (MMU), cache unit (CU), memory controller (MC), the network (NI) and switch (SI) interfaces. Figure 2 on page 3 shows the organization of the Processor-Memory element/subsystem (PME). The MMU in this PME organization is responsible for virtual memory management and also controls the cacheability at a page level. The actual cache management is done by the CU. The CU supports a 32K-byte common instruction/data cache. This is a software-managed cache, that is: (1) software maintains cache coherence without hardware support; and (2) cacheability is defined at the page level, this information is stored along with the virtual memory management information. The NI is responsible for directing memory references to either its PME's memory or to another PME's memory across the network. This information is derived by the NI from the memory reference's address. More detailed descriptions of the PME operations are given in [BRAN85] and [RP3P88]

This system supports both private and shared addresses spaces. Virtual memory management is done across both these spaces. The translated real addresses can also be interleaved and hashed across the memory modules of the system through the network. The number of memory modules a real page is interleaved and hashed across is defined on a page basis by the system. The MMU controls interleaving by providing the interleave factor, while hashing is controlled by turning it ON/OFF for each PME by software. If hashing is turned ON, then all translated addresses are hashed. Addresses are hashed across the memory modules specified by the interleave factor. The NI actually executes this interleaving and hashing. More information on these memory operations is given in [PFIS85, RP3P88].

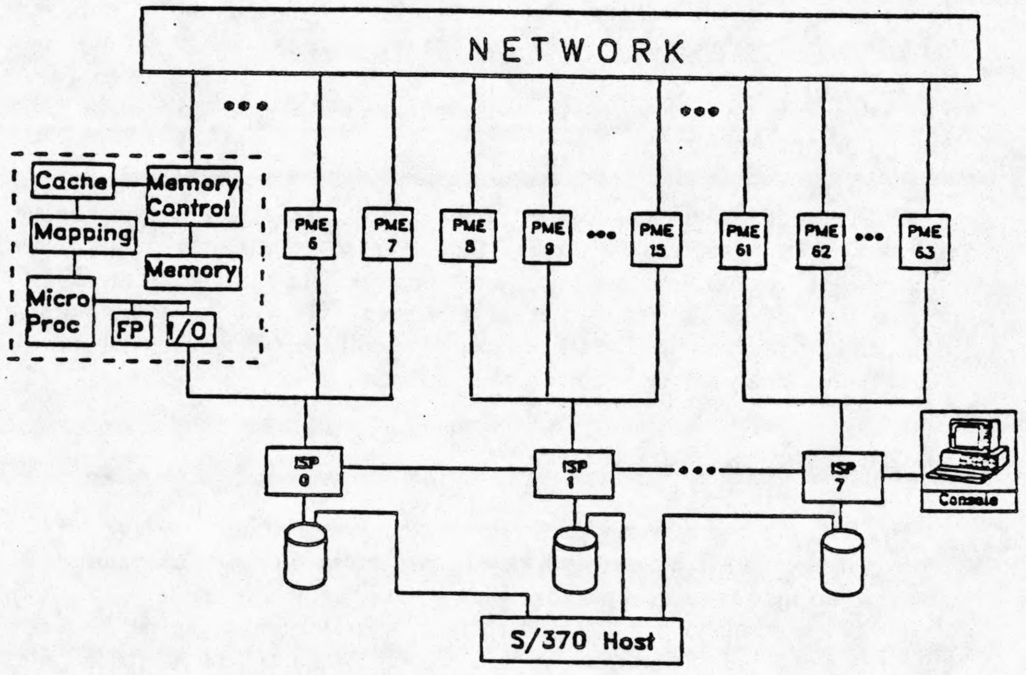
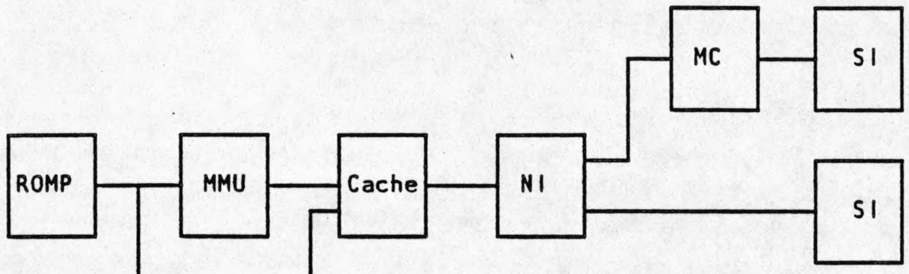


Figure 1. RP3 System Overview



Legend:

Address/Data Buses ———

ROMP	Processor
MMU	Memory Mapping Unit
Cache	Data and Instruction Cache
NI	Network Interface
MC	Memory Controller
SI	Switch Interface
FPU	Floating Point Unit
IOI	I/O Interface
PM	Performance Monitor

Figure 2. RP3 Processor-Memory Element (PME)

For RP3, testing issues were complicated by several system design/integration issues. First, the Level Sensitive Scan Design (LSSD) [EICH78] rules were only followed at the chip level. They were not followed at the card or subsystem level. Second, besides LSSD no other test support logic was provided in the design. The chips/cards did not provide any test points to interface test equipment. Therefore, the ability to control or observe a card or subsystem test was limited. This issue was aggravated further because it was not possible to connect any test equipment to the subsystems assembled in the RP3 frame. Finally, it was not possible to single cycle or multiple cycle the system.

Due to these limitations and because we wanted to support testing at system speed, we decided to functionally test the MMU. The chips used by the MMU were individually tested using scan-based tests, before they was integrated as one unit. The diagnostics to test the MMU were developed using the system's instruction set only. Only one set of diagnostics was developed. These diagnostics were used to debug the hardware in the lab. They were also used for system integration and system qualification.

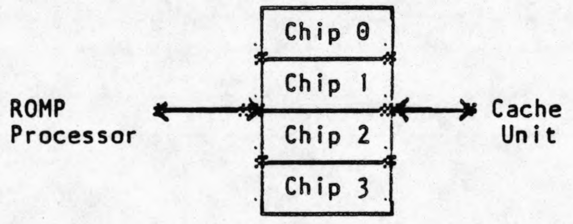
To develop these diagnostics, we started with the high level functional description of the MMU and the system. The description of the MMU's functions was in the system's Principles of Operations (POPs) manual. We supplemented this with discussions with the engineers whenever the POPs manual did not offer enough information. Only when timing sensitive or pattern sensitive faults were found during system debug in the lab, did details of the logic design have to be studied. Working with high level design information has the following advantages: (1) it verifies the functions in the design; (2) checks the correctness of the information in the POPs manual; and most of all (3) makes the diagnostics portable across technology upgrades. This last point is especially important because of the large effort usually invested in developing diagnostics.

Previous work in functional diagnostics has mainly been based on testing microprocessors. MMU testing issues have been discussed by Giles and Scheuer [GILE86]. Although their work also uses a structured testing approach similar to that suggested by FDM, their tests assume availability of test hardware in the design. Their methods also assume that automatic test equipment will be used to execute these tests. In the RP3 system, none of these assumptions were supported. Further, their MMU organization is different from RP3's.

3.0 MMU Organization

Each PME of the RP3 has a MMU. This MMU uses a 1 KByte direct-mapped translation look-aside buffer (TLB). The TLB is organized as 2 sets with 64 entries each, where each entry is 8 bytes. A random replacement policy is used for the TLB. The TLB can be managed either by software or via the hardware reload logic of the MMU. Besides doing the virtual-to-real translation, the MMU also checks the access protection and cacheability of the memory reference. The memory interleaving factor is also provided by the MMU.

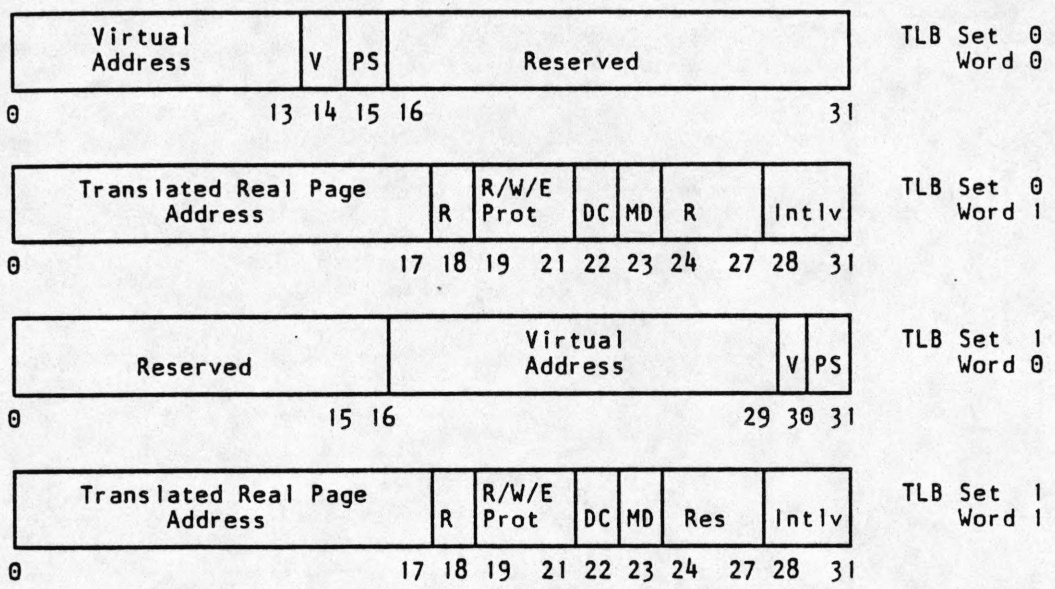
The MMU is physically organized as shown in Figure 3. The MMU unit is a byte-sliced design, which consists of four copies of a MMU chip. Each copy of this chip operates on its own byte of data. Special I/O lines have been defined on this chip to coordinate the MMU TLB look-up and control between the four chips. Each chip is informed about the byte it controls during processor initialization time. All busses shown in Figure 3 are 32-bit wide and have parity checking/generation on each byte.



Busses: 32 data bits plus 4 parity bits

Figure 3. MMU Organization

The layout of the TLB entries for sets 0 and 1 is shown in Figure 4 on page 5. Each entry is two words wide. The contents of the TLB entries are described by the legend of Figure 4 on page 5. Each entry points to a 16K Byte page in memory. Software has full access to the TLB entry words, just like main storage. This direct TLB addressing is used for software management of the TLB.



Legend:

DC	– Data cacheability bit
Intlv	– Interleave amount for the page
MD	– Marked data bit
PS	– Problem/Supervisor State
R	– Reserved
Res	– Reserved
R/W/E	– Read/write/execute protection information
TLB	– Translation Look-aside Buffer
V	– Page table valid bit

Figure 4. Format of a Translation Look-aside Buffer (TLB) Entry

The MMU has two modes of operation:

1. The translated mode uses the TLB to map virtual addresses into real addresses. In this mode, the real addresses can be interleaved and hashed across the main memory modules of the system.
2. The untranslated mode allows the processor to access the storage directly. In order to allow a PME to directly address the memory attached to it (see Figure 2 on page 3), the MMU exclusive-ORs this address with the processor's network address. This allows the PME's NI to direct the memory requests correctly. The resulting lower-order addresses are mapped to the memory attached to the processor.

An overview of these memory translation operations is given in Figure 5 on page 6.

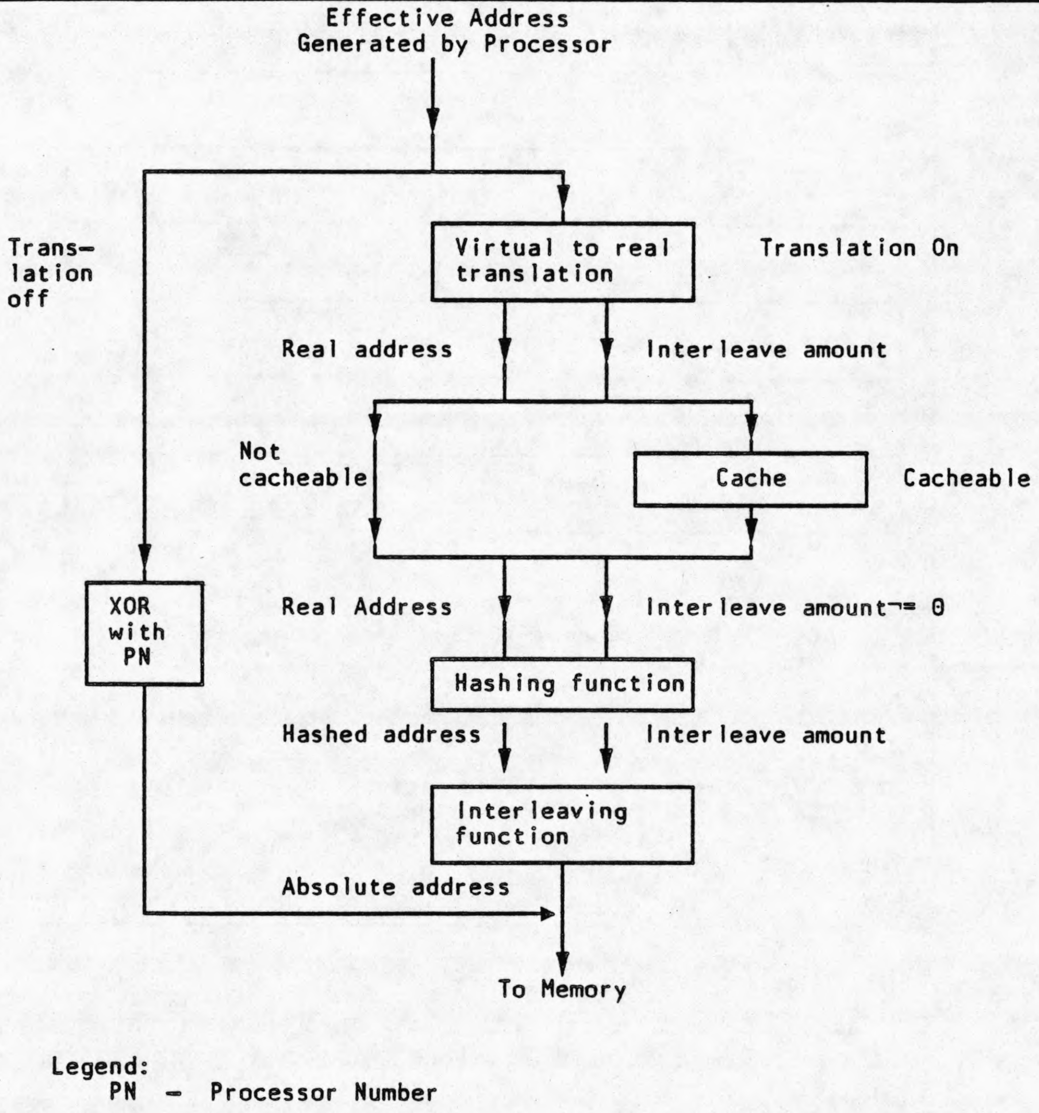
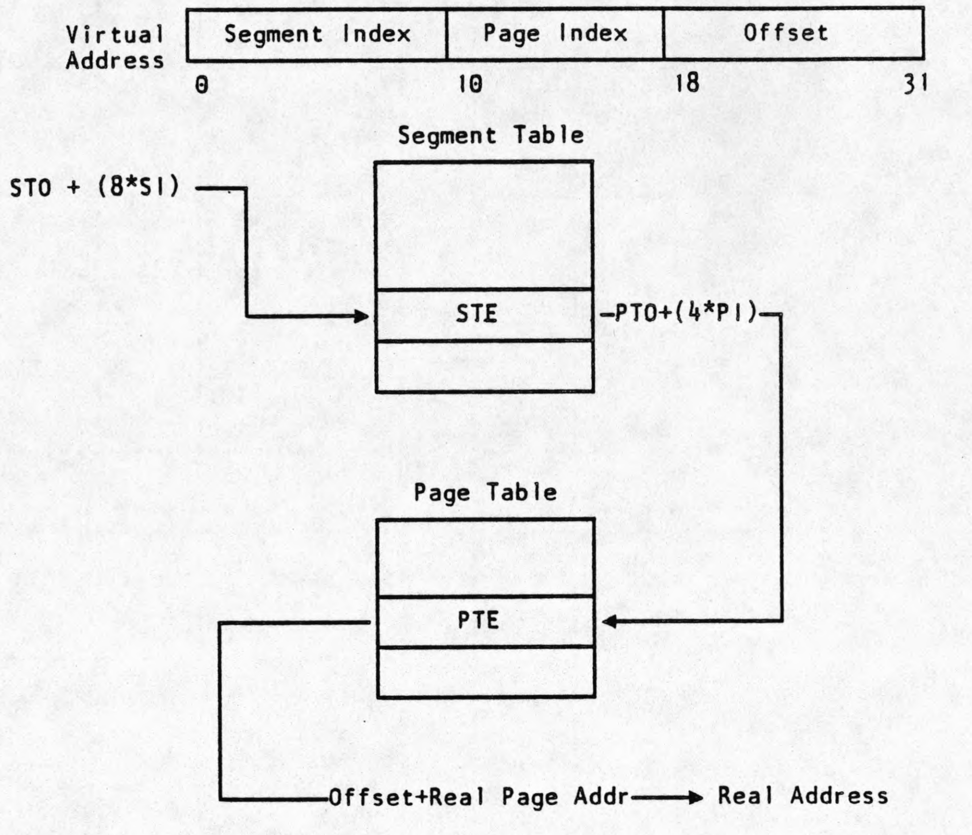


Figure 5. Overview of Memory Translation

During the translation of a virtual address, the MMU uses the protection information in the TLB entry to determine if the type of access requested is allowed for the virtual page. If the page is protected against this type of request, then an exception is raised. An exception is also raised when the MMU detects that a virtual address is mapped to both the sets of the TLB entry. These exceptions are serviced by software.

If a TLB hit takes place and the virtual address translation is done without any exceptions being generated, then the MMU passes the appropriate information to the cache unit. This information includes the Real Page Address, PS, DC, MD and INTLV fields in the TLB entry. If a TLB miss occurs, then a required TLB entry is selected, initialized appropriately and then the processor's request is tried again. This TLB management can be done by either software or MMU hardware.

If reloading is enabled and a TLB miss has occurred, then the MMU selects the TLB set to be used. If both sets are used or available, a random policy is used to select the set. The virtual memory tables in the main memory are then accessed to initialize the selected TLB entry. After the TLB entry is reloaded, the processor's request is retried and normal address translation is continued. An overview of this table access is shown in Figure 6 on page 7 More detailed information is given in [RP3P88].



Legend:
- STD – Segment Table Descriptor
- STO – Segment Table Origin
- STE – Segment Table Entry
- SI – Segment Index
- PTO – Page Table Origin
- PTE – Page Table Entry
- PI – Page Index

Figure 6. Virtual Memory Table access during TLB Hardware Reload

4.0 TLB Array Topology

The TLBs are implemented using 16x18 bit static embedded RAM arrays on the MMU chips. Six embedded RAM arrays per chip are used. Each 16x18 RAM array is physically addressed as two 16x9 RAMs. In this paper, we refer to these RAM portions as upper and lower. Each word in these RAMs contain 8 bits of data and a parity bit. The layout of the TLB using these RAMs is shown in Figure 7 on page 8. It should be noted that only half of TLB word-0 is useable, therefore RAM locations to store only these bits are provided.

	Word 0		Word 1			
TLB Set 0 64 entries	16x9 RAM #1 chip 0 Upper	16x9 RAM #1 chip 1 Upper	16x9 RAM #3 chip 0 Upper	16x9 RAM #3 chip 1 Upper	16x9 RAM #3 chip 2 Upper	16x9 RAM #3 chip 3 Upper
	16x9 RAM #1 chip 0 Lower	16x9 RAM #1 chip 1 Lower	16x9 RAM #4 chip 0 Upper	16x9 RAM #4 chip 1 Upper	16x9 RAM #4 chip 2 Upper	16x9 RAM #4 chip 3 Upper
	16x9 RAM #2 chip 0 Upper	16x9 RAM #2 chip 1 Upper	16x9 RAM #5 chip 0 Upper	16x9 RAM #5 chip 1 Upper	16x9 RAM #5 chip 2 Upper	16x9 RAM #5 chip 3 Upper
	16x9 RAM #2 chip 0 Lower	16x9 RAM #2 chip 1 Lower	16x9 RAM #6 chip 0 Upper	16x9 RAM #6 chip 1 Upper	16x9 RAM #6 chip 2 Upper	16x9 RAM #6 chip 3 Upper

	Word 0		Word 1			
TLB Set 1 64 entries	16x9 RAM #1 chip 2 Upper	16x9 RAM #1 chip 3 Upper	16x9 RAM #3 chip 0 Lower	16x9 RAM #3 chip 1 Lower	16x9 RAM #3 chip 2 Lower	16x9 RAM #3 chip 3 Lower
	16x9 RAM #1 chip 2 Lower	16x9 RAM #1 chip 3 Lower	16x9 RAM #4 chip 0 Lower	16x9 RAM #4 chip 1 Lower	16x9 RAM #4 chip 2 Lower	16x9 RAM #4 chip 3 Lower
	16x9 RAM #2 chip 2 Upper	16x9 RAM #2 chip 3 Upper	16x9 RAM #5 chip 0 Lower	16x9 RAM #5 chip 1 Lower	16x9 RAM #5 chip 2 Lower	16x9 RAM #5 chip 3 Lower
	16x9 RAM #2 chip 2 Lower	16x9 RAM #2 chip 3 Lower	16x9 RAM #6 chip 0 Lower	16x9 RAM #6 chip 1 Lower	16x9 RAM #6 chip 2 Lower	16x9 RAM #6 chip 3 Lower

Figure 7. TLB Array Layout

5.0 Overview of MMU Diagnostic Methodology

Using the FDM test approach, the MMU's functions were partitioned into smaller functional blocks. Separate tests were developed for each of these functional blocks. Each test concentrates on only testing its particular functional block and assumes that all other functional blocks are fault free. Dividing the module's functions also tends to divide the hardware into individually testable blocks. This reduces the test complexity. In order to generate the tests, a block's function and fault model is defined, the test patterns are derived and, finally, the instruction sequence needed to test the function is identified. The tests are then combined into a diagnostics program.

The MMU's functions were partitioned as shown in Figure 8 on page 9. Tests were derived for each of these functional blocks. It should be noted that no separate block was defined for the interleave logic. This is because the MMU only stores the interleave factor, the actual interleave function is executed by the PME's NI. This interleave factor is sent to the NI along with the translated address. Therefore, interleaving was tested by the NI diagnostics.

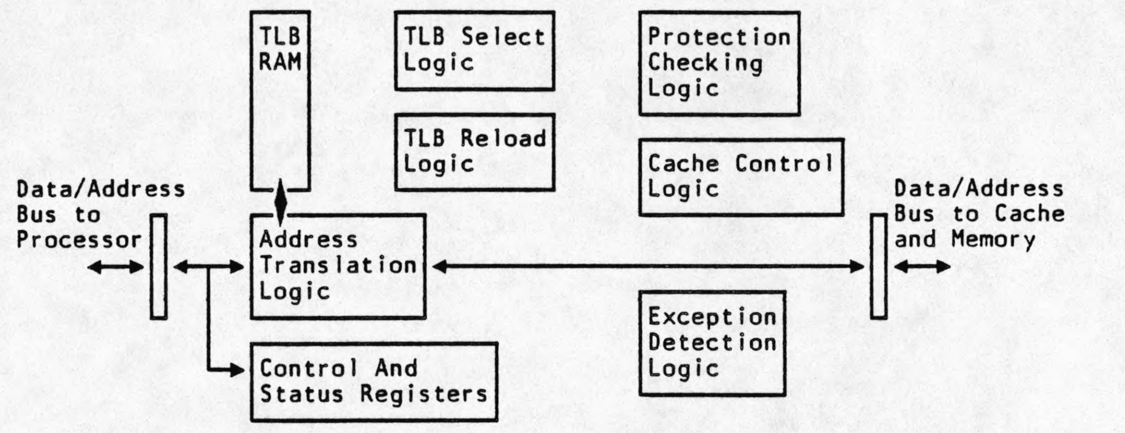

TLB RAM
TLB Select Logic
Protection Checking Logic
TLB Reload Logic
Cache Control Logic
Data/Address Bus to Processor
Address Translation Logic
Data/Address Bus to Cache and Memory
Exception Detection Logic
Control And Status Registers

Figure 8. RP3 MMU Functional Overview

The test patterns used by the tests are shown in Figure 9 on page 10. These patterns can detect stuck-at and coupling faults within a byte and across bytes of a word [THAT79]. Since each byte of the TLB is stored in a different embedded RAM, we only need to use the first set of test patterns. In order to test the word wide data/address busses of the MMU, both the test pattern sets were used. To test the parity bits/lines the parity test patterns were derived. The execution order of these test patterns (shown in Figure 9 on page 10) detects transition faults in the memory arrays.

These patterns can be derived as follows: start with a byte or word pattern of all zeros. This is the first pattern. Next, divide the bits in half, set the upper bits to zero and the lower bits to one. This gives the next pattern. To generate the third pattern, divide the bits into fourths, setting each alternating group of bits to zeros and ones. Continue this subdividing of bits until the alternating zero and one bit pattern is derived. Then, for each of these patterns compute their complementary pattern to complete this procedure.

```
        0000 0000
        FFFF FFFF        Data Patterns (Even Parity)
        0F0F 0F0F
        F0F0 F0F0

        007F 7F00        Parity Patterns (Odd Parity)
        7F00 007F

        3333 3333
        CCCC CCCC        Data Patterns (Even Parity)
        5555 5555
        AAAA AAAA

   a) Test Patterns for intra-byte coupling


        0000 FFFF        Data Patterns (Even Parity)
        FFFF 0000

        7F00 0000
        007F 0000        Parity Patterns (Odd Parity)
        0000 7F00
        0000 007F

        00FF 00FF        Data Patterns (Even Parity)
        FF00 FF00

   b) Test Patterns for inter-byte coupling
```

Figure 9. Test Patterns

6.0 Fault Model and Test Generation

While generating the tests, our aim was to detect permanent

1. Stuck-at faults
2. Pair-wise coupling faults
3. Transition faults
4. Functional faults

Due to the limited ability to control or observe at the instruction level, some tests were unable to detect some of these faults. In the following sections, test generation for each of the MMU functional blocks is discussed and where appropriate limitations on using this test strategy is identified. In the discussion below, it is assumed that caching, interleaving, hashing and MMU hardware reload are turned OFF, unless explicitly stated otherwise.

6.1 TLB Arrays

Since the TLB entries can be accessed directly by the processors, we can test the memory arrays as simple RAMs, ignoring the information they store. The TLB entries are mapped to the processor's I/O address space [RP3P88]. Therefore, they can be accessed using the IOR and IOW instructions of the ROMP [ROMP86].

It has been shown by [NAIR78] that any failure in a RAM is equivalent to failures in the memory cell array. This significantly reduces the RAM fault model. Several methods have been defined to test RAMs [ABAD83] and can be classified as those that detect stuck-at faults and those that detect coupling faults. We prefer the methods defined in [NAIR79] and [NAIR78] because they can be used for any RAM design. This is important for the functional diagnostics approach suggested here. The complexity of these RAM tests are:

- 5N-2 for the MATS+ [NAIR79] test, where "N" is the number of words in the RAM. Only stuck-at faults can be detected.
- 30n for the [NAIR78] test, where "n" is the number of bits in the RAM. Both stuck-at and coupling faults can be detected.

We selected the second method to test the RP3 TLB arrays because it provides more fault coverage. The basic algorithm is shown in Figure 10. We modified it to use the test patterns defined in Figure 9 on page 10. The algorithm is executed with each pair of test patterns.

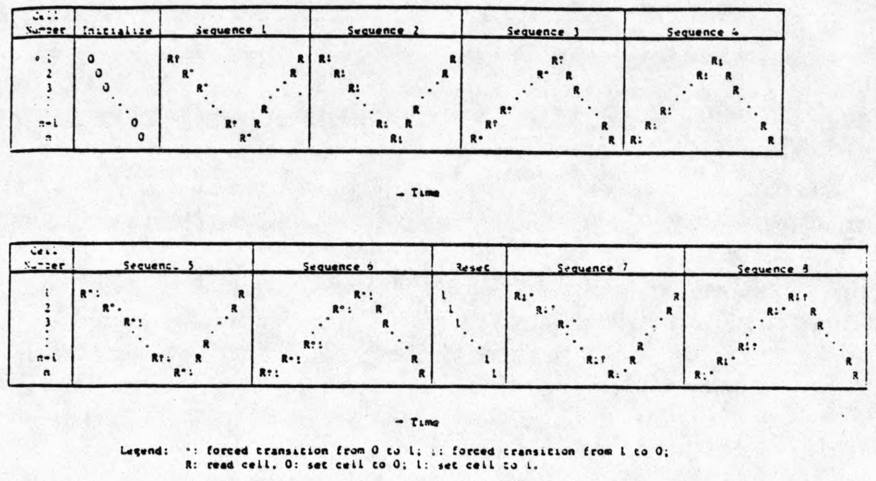

Cell Number
Initialize
Sequence 1
Sequence 2
Sequence 3
Sequence 4
Sequence 5
Sequence 6
Reset
Sequence 7
Sequence 8
→ Time
Legend: °: forced transition from 0 to 1; :: forced transition from 1 to 0;
R: read cell, 0: set cell to 0; 1: set cell to 1.

Figure 10. NTA Algorithm [NAIR78]

6.2 Address/Data Busses

The MMU is connected via a set of 32-bit address/data busses, as shown in Figure 3 on page 4. In this test we check the bus connecting the TLB arrays to the processor and the cache unit. Although the TLB array test uses this bus, it does not fully test it, because of the test patterns it uses. In order to completely test this bus the inter-byte coupling test patterns are used. This test selects a TLB word and writes these patterns to it. After each write the TLB word is read. If it does not match the test pattern, then an error is flagged. The bus that is used to communicate data from the processor to the cache and memory is tested by our main memory diagnostics.

6.3 Translation Logic

The untranslated mode is checked by executing some writes and reads to the memory of the processor. If these operations cannot be executed, an error is flagged. This only tests the exclusive-OR operation (see Figure 5 on page 6) needed to address the memory attached to the processor. In order to verify that the other memory modules of the system can be accessed, similar memory accesses are made to them in our network diagnostics.

In order to test the translation mode, the test patterns shown in Figure 9 on page 10 were used as virtual addresses. These patterns help check the TLB hit detecting logic. TLB set-0 was chosen and the entries, corresponding to the virtual pages of these virtual addresses, were initialized to map them to real pages residing in the PME's memory. The protection bits were set to allow read and write access. Test patterns were then written to these virtual pages. The results were verified by reading these locations in translated and untranslated modes. If the patterns written did not match the patterns read, an error was flagged. A similar test was executed using TLB set-1. In order to test the TLB hit detecting logic, these tests were done in both supervisor and problem (user) modes.

6.4 Hardware TLB Reload

This test checks to see if the TLBs can be loaded by the MMU hardware when a TLB miss occurs. In order to test the full function of this logic, reloading is checked under the following conditions: (1) the entries in both the sets are invalid; (2) only one set's entry is invalid and the other is allocated to another virtual page; and (3) the entries of both sets are allocated to other virtual pages. In addition, the reload function is checked when a valid TLB entry exists for the virtual page being referenced. This test checks if the hardware is executing unnecessary reloads.

The test patterns shown in Figure 9 on page 10 are used as the virtual addresses for this test. The virtual memory page table entries corresponding to the virtual memory pages for these addresses are appropriately set. They are mapped to real pages residing within the PME's memory. The TLB hardware is also initialized and hardware reloading is enabled. Then the TLB entries corresponding to these virtual pages are set according to the condition (mentioned above) being tested. A translated write to these virtual pages is then executed. After each write, the targeted main memory word is read, using the untranslated mode, to verify this write. The TLB entries are also read to verify that they are valid and are initialized with the correct information. A mismatch in either of these two verification operations is flagged as an error.

In order to determine if the hardware is executing unnecessary reloads, a virtual page is selected and its corresponding TLB entry is initialized to translate this page. Then the virtual memory page table entry for this virtual page is modified to point to a different real page than that pointed to by the TLB entry. TLB reloading is enabled and a translated write to this virtual page is executed. To verify this write, the word in the real page indicated by the TLB entry is read, using the untranslated mode. The TLB entry is also read to see if its contents have changed. If either of these verification operations indicates a mismatch, then an error is flagged.

6.5 TLB Replacement Policy

This replacement policy is used by the TLB reload hardware. A random policy is used by the MMU. In order to diagnose this policy, a test to verify that the MMU does not always select the same set is executed. If it does, then an error is flagged. This test can be done during the hardware reload testing and was integrated with it.

6.6 Protection Logic

This logic verifies if the memory operation requested by the processor is allowed for the virtual page. To test this logic, a virtual page is selected and its TLB entry is initialized to map this page to a real page in the PME's memory. First, the read, write and execution protection bits are set to zero. The real page in memory is then initialized using untranslated mode addressing. A translated mode reference of each of these types is then executed. If an exception is not generated for all three of these references, an error is flagged. Next the read protection bit is set and similar translated accesses are executed. Only the read access should not generate an exception. Errors are flagged if an exception is generated for the read request and not generated for the other two requests. An error is also flagged if an unexpected pattern is read by this read operation. Tests for the other combinations of protection bits are executed similarly. These tests were executed under both supervisor and problem modes.

During TLB hardware reloads, the protection bits for the virtual page's entry in the Segment table and the Page table are ANDed together to produce the protection bits for the TLB entry. This operation also needs to be tested. Since these bits are stored, accessed and used in parallel, we test them for stuck-at faults and coupling faults between these bits. The 8 protection bit settings shown in Figure 11 are used to set these bits in the Segment and Page table entries. These test patterns are the minimal set of patterns needed to detect pair-wise coupling faults [THAT79]. If all possible bit combinations for these pair of protection bits were used, then we would have had 64 test patterns. This test is executed in a similar fashion as the hardware TLB reload test.

```
Segment entry:          R=0, W=0, E=0
Page entry:             R=0, W=0, E=0
Expected results:       R=0, W=0, E=0

Segment entry:          R=1, W=1, E=1
Page entry:             R=1, W=1, E=1
Expected results:       R=1, W=1, E=1

Segment entry:          R=0, W=1, E=1
Page entry:             R=0, W=1, E=1
Expected results:       R=0, W=1, E=1

Segment entry:          R=1, W=0, E=0
Page entry:             R=1, W=0, E=0
Expected results:       R=1, W=0, E=0

Segment entry:          R=1, W=0, E=1
Page entry:             R=1, W=0, E=1
Expected results:       R=1, W=0, E=1

Segment entry:          R=0, W=1, E=0
Page entry:             R=0, W=1, E=0
Expected results:       R=0, W=1, E=0

Segment entry:          R=0, W=0, E=0
Page entry:             R=1, W=1, E=1
Expected results:       R=0, W=0, E=0

Segment entry:          R=1, W=1, E=1
Page entry:             R=0, W=0, E=0
Expected results:       R=0, W=0, E=0
```

Figure 11. Protect Bit test cases

6.7 Cacheability Control

The information on the cacheability of a page is stored in the MMU's TLBs. The MMU passes this information to the cache with every translated address. The test to verify if this MMU information controls caching is part of the cache unit diagnostics [KUND88]. Therefore, we only test the setting of the cacheability bit during hardware TLB reload. Similar to the protection bits, the AND of the segment and page table data cacheability bit is used to set the bit in the TLB. The test for this function is similar to the write protection logic test. The difference here is that this test requires checking if the data was actually cached or not cached. All settings of the cacheability bits in the segment and page table entries were used as test patterns (i.e. 4 test patterns).

6.8 Control and Status Registers

If registers can be read and written directly, then they can be tested as memory locations using both sets of patterns shown in Figure 9 on page 10. But the RP3 MMU does not provide this symmetric read-write access to its control and status registers. Writing some of these test patterns to the control registers would place the machine into unusual states. Some of the desired bit combinations were illegal. Similarly, the status registers restricted the bit patterns they could hold.

Therefore, instead of a complete test, we exercised the MMU in each of the states that the control registers allowed. We insured that each bit had the desired affect. For status registers, the condition which set a status bit was forced and the register read and verified. However, some conditions can only appear if the hardware fails. Since no test support hardware was provided, these conditions could not be tested. Only those conditions that had not been checked in another test were tested here.

6.9 Exception Detection

The MMU detects a number of functional error conditions and reports them to the processor through a program check. A number of hardware error conditions are also detected and reported as program checks. In this test, each MMU functional error was asserted. If the program check occurred, then the test passed. If a program check did not occur, an error was flagged. The functional errors tested were:

- TLB double mapping errors - that is, a virtual address is mapped to both the sets of the TLB entry,
- TLB miss, and
- Protection violations.

For testing efficiency, other functional errors, which had been tested in another test, were not duplicated here. The hardware error conditions could not be tested due to lack of test support logic.

7.0 Experiences

Our goals, when we started this work, were to develop a set of diagnostics which could verify the function of the RP3 system. These diagnostics were supposed to be executed each time the machine was powered on and during periodic maintenance check points. A simple GO/NOGO answer at the end of their execution and some indication of which subsystem failed was considered to be acceptable. But as we defined the diagnostics strategy for the system, we found that we could detect and isolate the faults within the subsystems. Therefore, we aimed our efforts to diagnose faults at the module level.

During the diagnostics development, an attempt was made to derive the tests using only the functional information. For example, information given in the Principles of Operations (POPs) manual of the system. But we found that some hardware organization information was essential to define some of the tests. For example, the layout of the RAMs and information on the type of error checking method used by the hardware was needed.

These diagnostics have been used to debug the MMU in the lab and when integrated into the system. They have also been used to qualify the MMU cards on a volume basis. They helped detect design errors, hardware faults, and errors in the documentation in the POPs manual. They have also detected all known timing problems in the PME design. An interesting design problem that was detected was that the LRU policy designed for TLB replacement was incorrect. The diagnostics showed that the MMU logic actually used a "somewhat random" policy. The diagnostics also discovered a discrepancy between the hardware and the POPs manual, concerning the setting of the cacheability information during hardware reload.

During our hardware debug experience in the lab we found that it was important to use patterns that asserted the parity bits and lines. Such errors and some of the pattern sensitive timing errors were identified in the lab while executing system and application programs. The diagnostics had to be upgraded to detect them, after the underlying conditions to assert these errors were understood. The diagnostics were easy to upgrade due to the modular test development approach used.

It took about 4 man-months to design, develop and debug the MMU diagnostics. The diagnostics were written in ROMP assembly. They were debugged using the RP3 instruction level simulator on an IBM 3084 system.

8.0 Summary and Conclusions

Our experience shows that very effective diagnostics can be developed using the system's instruction set. Using the FDM approach discussed in the paper, such diagnostics can be developed rapidly using a small number of people. They can be used to test, at system speed, the hardware in the lab and when integrated into the system. Only one set of diagnostics needs to be developed. These diagnostics are portable across technology upgrades of the system.

Acknowledgments

We would like to acknowledge the RP3 project team for their suggestions and support. We would like to thank Ton Ngo for his assistance with the MMU logic.

References

[PFIS85] Pfister, G.F., Brantley, W.C., George, D.A., Harvey, S.A., Kleinfelder, W.J., McAuliffe, K.M., Melton, E.A., Norton, V.A. and Weiss, J.; "The IBM Research Parallel Processor Prototype (RP3): Introduction and Architecture"; Proc. of the International Conference on Parallel Processing; St. Charles, IL; August 1985, pp 764-771.

[RATH88] Rathi, B.D., Kunda, R.P. and Hamilton, J.; "A Functional Diagnostics Methodology"; Proc. of the IBM TEST ITL; Burlington, VT; Fall 1988.

[ROMP86] "IBM RT Personal Computer Technology"; IBM Form No. SA23-1057; IBM Austin, Texas; 1986.

[BRAN85] Brantley, W.C., McAuliffe, K.P. and Weiss, J.; "RP3 Processor-Memory Element"; Proc. of the International Conference on Parallel Processing; St. Charles, IL; August 1985, pp 764-771.

[RP3P88] "The RP3 Principles of Operation"; RP3 Project; IBM, T. J. Watson Research Center, Yorktown Heights, NY; 1988.

[EICH78] Eichelberger, E.B. and Williams, T.W.,; "A Logic Design Structure for LSI Testability"; J. Design Automation and Fault-Tolerant Computing; Vol. 2, No. 2; May 1978; pp 165-178.

[ABAD83] Abadir, M.S. and Reghbati, H.K.; "Functional Testing of Semiconductor Random Access Memories"; Computer Surveys; Vol. 15, No. 3; September 1983; pp 175-198.

[NAIR78] Nair, R., Thatte, S.M. and Abraham, J.A.; "Efficient Algorithms for Testing Semiconductor Random Access Memories"; IEEE Trans. Computers; Vol. 27, No. 6; June 1978; pp 572-576.

[NAIR79[Nair, R.; "Comments on an Optimal Algorithm for Testing Stuck-at Faults in Random Access Memories"; IEEE Trans. Comput; Vol. 28, No. 3; March 1979; pp 258-261.

[THAT79] Thatte, S.M.; "Test Generation for Microprocessors"; Ph.D. Dissertation; University of Illinois at Urbana-Champaign; Urbana, IL; May 1979.

[GILE86] Giles, G. and Scheuer, K.; "Testability Features of the MC6885 PMMU"; Proc. of the International Test Conference; September 1986; pp 408-411.

[KUND88] Kunda, R.P., Hamilton, J., Lee, D. and Rathi, B.D.; "A Cache Diagnostics Method"; Proc. of the IBM Fall TEST ITL; Burlington, VT; September 1988.