December 1988

UILU-ENG-88-2267
CSG-97

COORDINATED SCIENCE LABORATORY
College of Engineering

A CACHE DIAGNOSTICS METHOD

Ramachandra P. Kunda
Jeffrey Hamilton
Dongjae Lee
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-88-2267 (CSG-97)	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/SRC/DARPA
6c. ADDRESS (City, State, and ZIP Code) 1101 W. Springfield Ave. Urbana, IL 61801	7b. ADDRESS (City, State, and ZIP Code) IBM: Yorktown Heights NY 10598 SRC: Research Triangle Park, NC 27709 (over)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION IBM/SRC/DARPA — 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER IBM 1249006; DARPA N00039-87-C-0122; 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)
A Cache Diagnostics Method

12. PERSONAL AUTHOR(S)
Kunda, Ramachandra P.; Hamilton, Jeffrey; Lee, Dongjae; Rathi, Bharat Deep

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM ______ TO ______	December 1988	18

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 cache of a computer system. This method was used to develop the cache diagnostics for the Research Parallel Processor Prototype (RP3), a highly parallel computer that is being prototyped at the Watson Research Center. The RP3 has a 32 KBytes "write-through" direct mapped cache architecture. This cache is used to cache both instructions and data. This cache supports both a random replacement policy and also a SANF (store-allocate-non-fetch) cache management policy designed especially for RP3. Besides the traditional caching functions, this cache also supports new features like temporarily cacheable data types and single instruction invalida-

(over)

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

DD Form 1473, JUN 86 Previous editions are obsolete. SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

UNCLASSIFIED

7b. Continued

DARPA: Arlington, VA 22209

19. Abstract (continued)

tion of the whole cache or the lines of a specified page.

The RP3 cache is a multiple chip design. Conventional scan-based test methods could not be used to test, because the LSSD rules were only followed at the chip level. The functional diagnostics approach suggested in this paper diagnose the cache via the system's instruction set. This method uses a divide and conquer philosophy to reduce the diagnostics complexity and size. We have used this diagnostics methodology to develop diagnostics for the RP3 cache. Besides being used in the system, these diagnostics are also being used by the engineers in the lab. They have allowed us to test the cache functionally and detect faulty hardware and timing problems in the logic.

A CACHE DIAGNOSTICS METHOD

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

Jeffrey Hamilton
Dongjae Lee
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 cache of a computer system. This method was used to develop the cache diagnostics for the Research Parallel Processor Prototype (RP3), a highly parallel computer that is being prototyped at the Watson Research Center. The RP3 has a 32 KBytes "write-through" direct mapped cache architecture. This cache is used to cache both instructions and data. This cache supports both a random replacement policy and also a SANF (store-allocate-non-fetch) cache management policy designed especially for RP3. Besides the traditional caching functions, this cache also supports new features like temporarily cacheable data types and single instruction invalidation of the whole cache or the lines of a specified page.

The RP3 cache is a multiple chip design. Conventional scan-based test methods could not be used to test, because the LSSD rules were only followed at the chip level. The functional diagnostics approach suggested in this paper diagnose the cache via the system's instruction set. This method uses a divide and conquer philosophy to reduce the diagnostics complexity and size. We have used this diagnostics methodology to develop diagnostics for the RP3 cache. Besides being used in the system, these diagnostics are also being used by the engineers in the lab. They have allowed us to test the cache functionally and detect faulty hardware and timing problems in the logic.

1.0 Introduction

This paper describes a Functional Diagnostics Method (FDM) to test the cache of a computer system. We are using this method to develop the cache diagnostics for the Research Parallel Processor Prototype (RP3), a highly parallel computer that is being built at the Watson Research Center. The RP3 is architected to have 512 processors, 2 GBytes of memory, 192 MBytes/sec I/O and an interconnection network that has a peak bandwidth of 13 GBytes/sec. A prototype is being built which has 64 processors. The organization of a 64-processor RP3 system is shown in Figure-1. The main motivating reason behind building this machine is to conduct research in the highly parallel processing area. In order to facilitate this research, the RP3 architecture supports many interesting features. For example, the machine can support both the shared memory and distributed memory models of parallel computing. It also supports a user controllable cache. 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 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 data. On each memory reference, the CU decides to cache the information, only if the MMU indicates that the page is cacheable. A more detailed description of the PME operations is given in [BRAN85, RP3P88]

In order to build any system successfully, a method to test the system must be defined. One or more methods may be needed to test the system at all levels of integration. That is, different methods may be needed for chip, card, subsystem, and system level testing. These test methods must not only be able to diagnose the system well (in terms of fault coverage), but they should be able to do this fast (in terms of time). For parallel processor systems which have a large number of subsystems, we find that these test requirements are even more stressed. For the RP3 system, we tested the chips using scan-based tests. The other levels of integration were tested using the Functional Diagnostics Method (FDM) discussed in [RATH88]. The FDM approach uses the system's instruction set to test the system.

In this paper, we discuss the use of FDM to test RP3's cache unit. Using this approach allowed us to develop the cache diagnostics quickly and use these diagnostics to test the CU beyond the chip level. Our experience with this approach has shown that these diagnostics can detect functional design errors, logic errors and some timing errors. They can even be used to qualify the cards and subsystems on a volume basis. In fact, these diagnostics have been used for system test, integration, and qualification of the RP3.

The paper is organized as follows: in the next section, our motivation to use the FDM approach to test the cache is discussed. The RP3 cache unit organization is described in section-3 and the cache memory topology in section-4. In section-5, an overview of the cache unit diagnostic method is given, while the fault model and test generation are described in section-6. The test patterns used by these tests are presented in section-7. Finally, the last two sections discuss our experience with these diagnostics and our conclusions.

2.0 Motivation

Test methods based on Level Sensitive Scan Design (LSSD) [EICH78] are often used to test logic chips. LSSD provides better control and observation during testing. Scan-based test methods have also been extended to test cards and to do system integration. Although these test methods are effective, they are also time consuming because of the large number of test patterns used. In systems like the RP3, which have a large number of components, using scan based test methods at all levels can be a formidable task. Therefore, other test techniques must be considered.

For RP3, testing issues were complicated further by several other system design/integration issues. First, the LSSD 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 and observe during card or subsystem testing 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 use functional testing to test the cache unit. The chips used by the cache unit were individually tested using scan-based tests, before they were integrated into a cache unit. The diagnostics to test the cache units were developed using the system's instruction set only. Only one set of diagnostics was developed. These were used to debug the hardware in the lab and during subsystem test, system integration, and system qualification.

To develop these diagnostics, we started with the high-level, functional description of the cache unit and the system. The description of the cache unit'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. Details of the logic design had to be studied only when timing sensitive or pattern sensitive faults were found during system debug in the lab. Working with this high level design information has the following advantages: (1) it verifies the functions of 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 important because of the large effort usually invested in developing diagnostics.

3.0 Cache Unit Organization

Each RP3 processor has a 32 KBytes, direct-mapped, "write-through" cache. A single cache is used to cache both instructions and data. This cache is organized as a 2-way set associative cache. Each set has 1K lines, where each line is 4 words (16 Bytes). This cache supports a random replacement policy and a SANF (store-allocate-non-fetch) cache management policy specific to RP3 [BRAN85]. In addition to the usual cache functions, this cache supports new features, like

- temporary cacheable data,
- single instruction invalidation of the whole cache, and
- single instruction invalidation of a set of lines belonging to a specific page.

Although the RP3 system contains mutiple processors, it does not have any hardware cache coherence mechanism. In this system, the software is responsible for keeping the cache coherent.

The cache unit is physically organized as shown in Figure-3. The cache control unit is a byte-sliced design, which consists of four copies of a cache control chip. Each copy of this chip operates on its byte of data. Special I/O lines have been defined on this chip to coordinate the cache 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. The directory (DIRMEM) and data memories (DATMEM) of the cache are packaged as separate memories.

The logical layout of the cache's directory and data memories is shown in Figure-4. The fields in the directory memory are:

KEY	the most significant 18-bits of the real (translated) address;
INTLV	the interleave factor for the memory reference's page;
MD	indicates that this line contains temporary data;
U/S	indicates if the memory page is a User or Supervisor page;
X	not used;
F	informs the cache that the data words associated with this line are currently being fetched from memory;
R0 to R3	indicate if the line's data word-0 to word-3 respectively are resident in the cache's data memory.

The CU generates the values of the F and R fields, while the contents of the other fields are supplied to the CU by the MMU.

The RP3 cache uses the real address to tag its data. The MMU unit does the virtual-to-real address translation for every memory reference. In order to reduce the time to determine a cache hit, the virtual address generated by the processor is simultaneously presented to both the MMU and the CU. The CU uses this address to fetch the directory entry and the associated word in the data memory (see Figure-5). The MMU, in the mean time, translates this address and determines if the memory reference is cacheable (in RP3 cacheability is defined on a page basis [RP3P88]). If it is cacheable, the MMU presents the most significant 18 bits of the translated address to the cache, along with the other relevant information. The cache uses this information to compare against the Key, Interleave, MD and U/S fields of the selected line. If they match and the required data word is resident (i.e. the respective R bit is set), then a cache hit is declared. The cache does this comparison for both its sets. Set-A is processed in the first half of a processor cycle, and Set-B in the second half. If a cache hit takes place, the cache transfers the data directly to the processor, in the following cycle. If the above match does not occur in either of the sets, then a cache miss is processed by the cache. If both the sets match, then a cache double mapping error is flagged.

If a cache miss occurs during a processor LOAD reference, the CU selects a set, initializes the directory entry, sets the F-bit and generates a set of two double word fetch requests to the memory, to fetch the cache line. These memory fetches are asynchronous. That is, the CU is not locked from receiving further requests. The F-bit is used to avoid generating an unnecessary cache line fetch, if one has been requested earlier. In the case of two successive memory accesses to the same cache line, the cache stops accepting any further requests until the first request is completed. The F-bit is reset when the data memory has been updated. If a cache miss occurs during a STORE reference, a set is selected and the directory is initialized. But due to the SANF policy, the cache does not prefetch the line's data, it only stores the STORE reference's data (in both DATMEM and the main memory) and sets the appropriate R-bit. The other R-bits are reset. A random selection policy is used to select the set, when both sets are used or available.

The RP3's architecture also supports read-modify-write (RMW) (e.g. Fetch&Add instructions [BRAN85]) and partial-word-store (PWS) type memory references, which are executed atomically. To guarantee this atomic execution, the cache implements these operations as follows: whenever the MMU indicates that a RMW is cacheable and there is a cache hit, the CU invalidates the word in the cache and passes the request to the main memory. The main memory atomically executes this request and the cache word is not updated. On a cache hit for a PWS request, only the bytes that need to be stored are updated in the cache. This operation is also "written-through" to the memory, which updates the memory atomically.

4.0 Cache Memory Topology

The DIRMEM and DATMEM memory arrays are implemented using 2Kx9 bit static memory modules. The layout of a word in each of these memories is identical, except for the number of words in each line. The DATMEM line consists of four 32-bit words, with parity per byte. There is one line per set. This gives us 32K bytes for cached data. The DIRMEM line consists of one 32-bit word, with parity per byte. There is one such line per set. This gives us 8K bytes for directory information.

The layout of the DATMEM array is shown in Figure-6. It should be noted that each byte of each word is assigned to a unique memory module. The parity for a byte is stored in the same memory module as its data. In this arrangement, the upper 1K addresses of each memory module are assigned to Set-A; while the lower 1K addresses are assigned to Set-B. The CU generates the address bit that selects the set. This enables it to control the set to be replaced. The line selection address bits and word selection bits are taken from the processor generated address. Note that at any time, only one word of one line is selected. The layout of DIRMEM's array is similar, but with one row of memory modules used.

5.0 Overview of Diagnostic Method

The basic idea in the FDM approach is to divide the functions of the module to be diagnosed, into smaller functional blocks. Tests are then written for these functional blocks. Dividing the module's function tends to divide the hardware into individually testable sections. During test generation for these functional blocks, all other functional blocks are considered to be fault free. 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.

In order to diagnose the cache unit, we can divide it into the following functional blocks:

- Memory arrays
  - DIRMEM arrays
  - DATMEM arrays
- Address/Data busses
- Comparator
- Error detection logic
- Status registers
- Control logic
  - Caching function
  - Cache Invalidation

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 and 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 cache functional blocks is discussed and where appropriate limitations on using this test strategy is identified. In the discussion below, it is assumed that the MMU is appropriately set to support these tests.

6.1 Memory Arrays

For testing, we treat the cache DIRMEM and DATMEM arrays as simple RAMs; ignoring the type of information they store. In order to test any RAM, it is important to be able to consistently access the same location for a given address. That is, it should support a one-to-one address-data mapping. The direct-mapped cache organization supports such a mapping and allows it to be done from the instruction level, because it guarantees that a memory reference is always associated with the same line and word of the cache. Notice that any associative cache organization will not support this one-to-one, address-data mapping at the instruction level. In order to provide this one-one control, some hardware support would be needed.

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 found the methods defined in [NAIR79] and [NAIR78] to be attractive, because they could be used for any RAM design. This is important for the functional diagnostics approach suggested here. The complexities 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 MATS+ to test the RP3 cache arrays because addressing limitations restricted us from using the coupling test. For example, the coupling test requires each bit of the RAM array be independently addressable. The cache arrays only support access to 9 bits (data plus parity) at a time.

The basic MATS+ algorithm is shown in Figure-7. For the RP3 cache we modified it in two ways:

1. Selecting the set to be used is not under processor control; it is done by the CU. A set is chosen randomly if both the sets are used or available for the addressed line. Although this modifies the ascending/descending addressing defined by the algorithm in Figure-7, it does not corrupt the correctness of the algorithm. This is because after the first initializing sequence, the sets are assigned randomly, but a logical ascending/descending addressing order is maintained for the rest of the algorithm.
2. To detect transition faults and some coupling faults, we used the test patterns defined in section-7. These test patterns detect coupling between any two bits of a cache byte. Inter-byte coupling does not need to be tested, because each byte of a cache word is packaged in separate memory modules (Figure-6). The algorithm is executed for each pair of test patterns.

6.1.1 DATMEM Arrays

The DATMEM array can store the contents of two RP3 memory pages (16K bytes per page). In order to test this array four virtual pages were selected. Two of them were made cacheable and were mapped to distinct real pages, while the other two were made non-cacheable and were mapped to each of the real pages. The cacheable pages were used for all the accesses required by the MATS+ algorithm. The non-cacheable pages were used to modify the main memory word with the complement of the test pattern after each cache write operation. This was done to guarantee that the contents of the cache array and the main memory array were different. This allowed us to verify whether the data was from the cache or main memory arrays. If a non-test pattern was obtained, then DATMEM was considered to be faulty for the addressed entry. If the main memory pattern was read, then the comparator, the array decoder or the DIRMEM was suspected to be faulty. If the test pattern was read, then the array was assumed to be functional for the entry.

6.1.2 DIRMEM Arrays

The DIRMEM arrays were more difficult to test, because they required a more elaborate virtual memory mapping than the DATMEM test. In order to use the test patterns defined in section-7, the tag information (Figures 4 and 5) for this memory needs to be appropriately set. For the RP3 cache, this information resides in the MMU. It was not possible to generate all the test patterns. For example, we could not set the F-bit, because of lack of direct control at the instruction level. The total amount of real memory in the prototyped system also restricted the Key field dependent test patterns.

Therefore, we modified the test patterns to enable us to use the MATS+ algorithm. For each pattern in a pair of test patterns, a bit in the Key field was identified and set to 0 for half of the cache lines and set to 1 for the other half. The remaining bits were set as required by the test pattern. Since the Key field identifies the real page being used, this modification identifies two cacheable memory pages. Two cacheable pages are needed to address all the DIRMEM words. The test setup required two virtual pages to be mapped to each of these real pages. One of these virtual pages was allowed to be cached, while the other was not. These four virtual pages were used as described in the DATMEM test.

The MATS+ algorithm was executed as explained in the DATMEM array test. The only difference was that the test patterns now defined the real pages to be selected and not the data to be written. Also, for each line, only one data word needed to be accessed. In order to identify a cache hit from a miss, a complementary pair of patterns was used for the data word pattern. One such pattern was written to the cache, while the other was used to update the memory word. If an expected cache hit or miss was seen, then the DIRMEM was considered to be functional. If an unexpected cache hit or miss took place, then the comparator or the DIRMEM array was suspect.

6.2 Address/Data Busses

The CU is connected via a set of 32-bit address/data busses, as shown in Figure-3. In this test, we only check the busses to the cache memories. The other busses are tested by our main memory diagnostics. Although the cache memory array tests use these busses, they do not fully test them, because the array test patterns only detect coupling between the lines belonging to a byte and not across bytes. The additional test patterns defined in section-7 need to be used to detect coupling between bytes. This test selects the appropriate virtual-real pages and uses them in a similar fashion as the DATMEM array test. The test patterns need to be asserted only once since no decoder logic is involved.

6.3 Comparator

In order to test the comparator for all possible stuck-at faults, we need to define more test patterns. To do this, we need to know the implementation details of the comparator. The test patterns derived for the comparator's logic can then be mapped to appropriate DIRMEM Tag values (Figure-5). A test sequence similar to the Bus tests can then be used to assert these Tag values.

For simplicity, we decided to avoid implementing the full comparator test for RP3. The CU's comparator is used to identify a cache hit (Figure-5). The comparator's cache hit function is tested during the DIRMEM array tests. In RP3's CU comparator test, we only tested the cache miss function. This test can be easily derived from the procedure defined for the DIRMEM array test.

6.4 Error Detection Logic

For data/address lines/storage, the main error detection method used by the CU is parity checking. Parity is checked for all information coming into the CU and generated for all information leaving the CU. Parity is also stored along with the cache information in both the memory arrays. The memory arrays and their bus parity logic is tested during their respective tests. This is done by selecting the test patterns carefully (discussed in section-7). The parity logic for the other paths are checked in a similar fashion in other diagnostics (e.g. main memory).

For the cache, we also need to detect double mapping errors. This error occurs when a line of a page is cached in both the sets. The RP3's CU does not provide any support for doing this intentionally; therefore, this logic cannot be tested using the FDM approach. Some testability support is needed to test this function.

6.5 Status Registers

The status register stores information regarding the errors seen by the CU. If these registers could be read and written, then a simple read-write test using the test patterns defined in section-7 would be sufficient. But since these registers could only be read, it was not possible to test them, since it was not possible to assert all the error conditions intentionally.

6.6 Control Logic

This section defines the procedures used to test the functions of the CU.

6.6.1 Caching Function

We need to test the following functions of the cache: the "write-through" policy, the "store-allocate-non-fetch" (SANF) policy, partial word stores (PWS) and read-modify-write (RMW) references. In all these cases, we use the "two virtual pages mapped to one real page" scheme used in the DATMEM array test.

We can test the "write-through" policy using one test pattern. A word in the test's virtual pages is selected. The cache entry and the memory location for this word are cleared and verified. Then a test pattern is written into the cache entry. If the cache and memory do not store the test pattern, then an error is flagged.

The SANF policy is similarly tested. A cache line is selected and a pattern containing all zeros is written in the line. A test pattern is then written to only the memory words corresponding to this line. Following this, a complementary test pattern is written to a word in the cache line and then the whole cache line is read. If any other word besides this modified one is non-zero, an error is flagged. Notice that the "write-through" test can be coupled with this test.

The RP3 architecture supports PWS references that operate on any byte and any half of a word. The cache implements these as cache store operations. But it only enables the required cache DATMEM array modules (see Figure-6). Therefore, in order to test the byte PWS function of the cache, we need 16 test patterns. We need to verify if the cache can selectively enable the memory modules one at a time (Figure-6). The half-word tests require 8 such test patterns. Testing is started by selecting a cache line and initializing it to zero. A test pattern is then written to only the memory words corresponding to this line. Following this, byte PWS are done to each of the 16 bytes of the lines. The test pattern used for each byte must be unique. At the end of the stores, the corresponding memory locations are read and verified. If they do not match the test patterns, an error is flagged. These memory locations are then cleared and the cache line is read. If the cache line contents do not match the test patterns, an error is flagged. The half-word PWS test is similar.

The RMW reference caching is tested by selecting a cache line and writing all zeros in its words. A test pattern is then written to only the corresponding words in memory. Following this, a cacheable RMW reference is made to a word of this line. The corresponding memory word is then read and checked to see if the memory executed the RMW correctly. This word in memory is then modified with its complemented data. The word accessed by the RMW reference is then read from the cache. A cache miss should normally occur; therefore, the main memory pattern should be read. If this does not happen an error is flagged.

6.6.2 Cache Invalidation

The CU supports the following cache invalidate functions: .line, page, full-cache and temporary-cacheable-data (TCD) invalidates. Special "opcodes" are defined for these functions by the RP3 architecture. A line is invalidated by specifying a word of the page that is mapped to this line. A page invalidation is done by specifying a word of the required page. Full-cache and TCD invalidation are done by executing the required operation code. It should be noted that these operations are User/Supervisor mode sensitive. For example, if the processor is executing in User mode when it requests a full cache invalidate, then only the User space lines are invalidated. Invalidation is done by resetting the residency bits in DIRMEM (Figure-4). This forces a cache miss on a following reference to an invalidated cache line. To test these functions, the "two virtual page mapped to one real page" scheme of the DATMEM array test is used.

To test line invalidation, a cache line is selected and initialized with a test pattern. The complementary test pattern is written in the corresponding memory words. The cache line is then invalidated. This invalidation is verified by reading the caching line and checking if the data read matches the pattern written in the memory. If they do not match, an error is flagged.

The page and full-cache invalidation is similarly tested. The difference here is that the whole cache is initialized and the whole cache is read and checked after invalidation. This verifies that the invalidation did not incorrectly invalidate a line (e.g. one belonging to a different page). Since these operations are User/Supervisor mode sensitive, these tests are executed in both these modes. Also the cache is initialized so that lines for both user and supervisor pages are resident in it.

Temporary, cacheable data is identified by the MMU. When the cache stores temporary cacheable data, it sets the MD bit in the DIRMEM (Figure-4). During TCD invalidation the cache only invalidates lines that have this MD bit set. The test to check this operation is similar to the full-cache invalidation test.

7.0 Test Patterns

The test pattern used to verify the memory arrays are shown in Figure-8(a). These test patterns can detect stuck-at and pair-wise coupling faults within the bytes of a word [THAT79]. They test both the data bits and the parity bits. The sequence in which these test patterns are used detects transition faults in memory arrays. For busses and registers that are a word wide, additional test patterns are used to check coupling between bytes. These additional test patterns are shown in Figure-8(b).

8.0 Experience

When we started this work, our goal was to develop a set of diagnostic programs which could verify the functions of the RP3. The diagnostics were supposed to be executed each time the machine was powered on. At the end of this execution a simple GO/NOGO answer and some indication of which subsystem failed was considered to be acceptable. But, while defining the diagnostics strategy for the machine, we saw that we could detect and isolate the faults within the subsystems. Therefore, we aimed our efforts to diagnose faults at the module level.

Although functional diagnostics have been used by many systems, we only found literature on testing microprocessors and random access memories functionally when we started this work. Recently, some work on testing the MC68030 caches was published [BAST87]. It is interesting to note that the approach used by [BAST87] is similar. The main difference in our approaches is that they use microcode to test the on-chip cache, while we use the system's instruction set. They also assume that automatic test equipment will be used to execute their tests. The cache architectures are also different.

While developing our diagnostics, we found that it was not always possible to test the cache as thoroughly as we desired. This was due to the lack of control, at the instruction level, over some logic of the cache. In these cases, we adapted the test to suit the ability to control the logic. For example, the directory memory test had to be adapted to suit the real memory available in the machine. We also noticed that the directory memory and the comparator tests could have been substantially simplified, if some testing support was provided by the cache design.

Even though we had to compromise on some of the tests, we found that the cache diagnostics were very effective. They have been used to debug the cache units in the lab and when integrated into the system. They have also been used to qualify the cache cards on a volume basis. The diagnostics have helped identify design errors and faulty hardware. They have also been able to detect the known timing errors in the cache design. These timing errors were pattern sensitive and were initially 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 timing errors were understood.

It took us about 3 man months to design, develop and debug these diagnostics. The diagnostics were written in ROMP assembly. They were debugged using the RP3 instruction level simulator on an IBM 3084 system.

9.0 Conclusions

Our experience with the diagnostics method proposed here has shown that cache units can be very effectively tested, at system speed, using a functional diagnostics approach. This method defines one set of diagnostics that can be used to test the hardware in the lab and when integrated into the system. These diagnostics have been able to detect permanent hardware faults and known timing errors in the cache logic. They have also been useful in verifying the information in the principles of operation type of documentation of the RP3 system.

This testing approach allows the diagnostics to be developed rapidly using a small number of people. The tests can be developed using only the system's instruction set. Although some control on testing is compromised at this level, we find the overall effectiveness is not lost. Further, this approach allows porting these diagnostics across technology upgrades of a system.

Acknowledgments

We would like to acknowledge the RP3 project team for their suggestions and support. We also would like to thank Ton Ngo for this assistance with the cache unit's 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.

[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.

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

[BAST87] Basto, L., Harrod, P. and Bruce, W.; "Testing the MC68030 Caches"; Proc. of the International Test Conference; Washington D.C.; September 1987; pp 826-833.

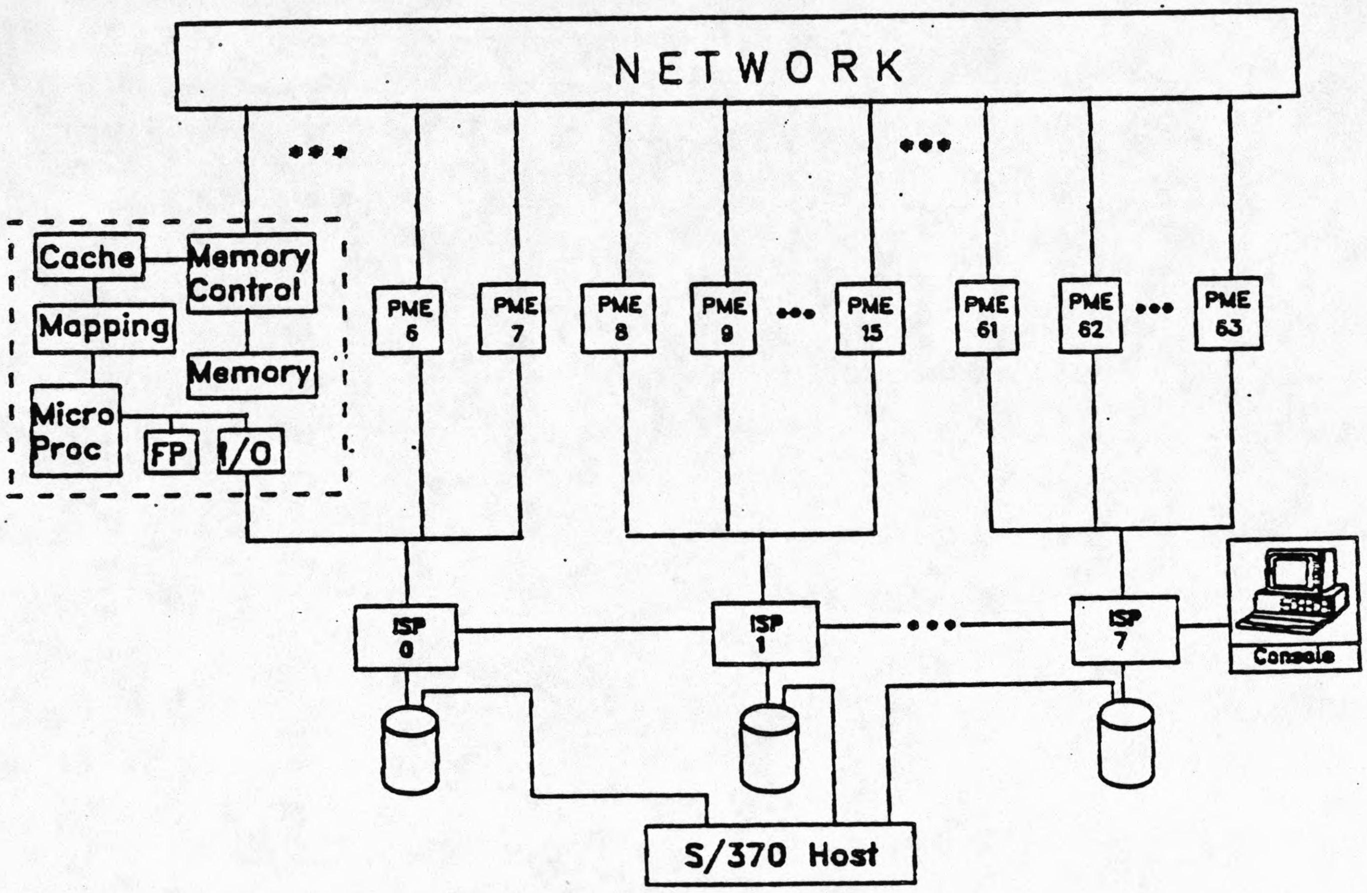
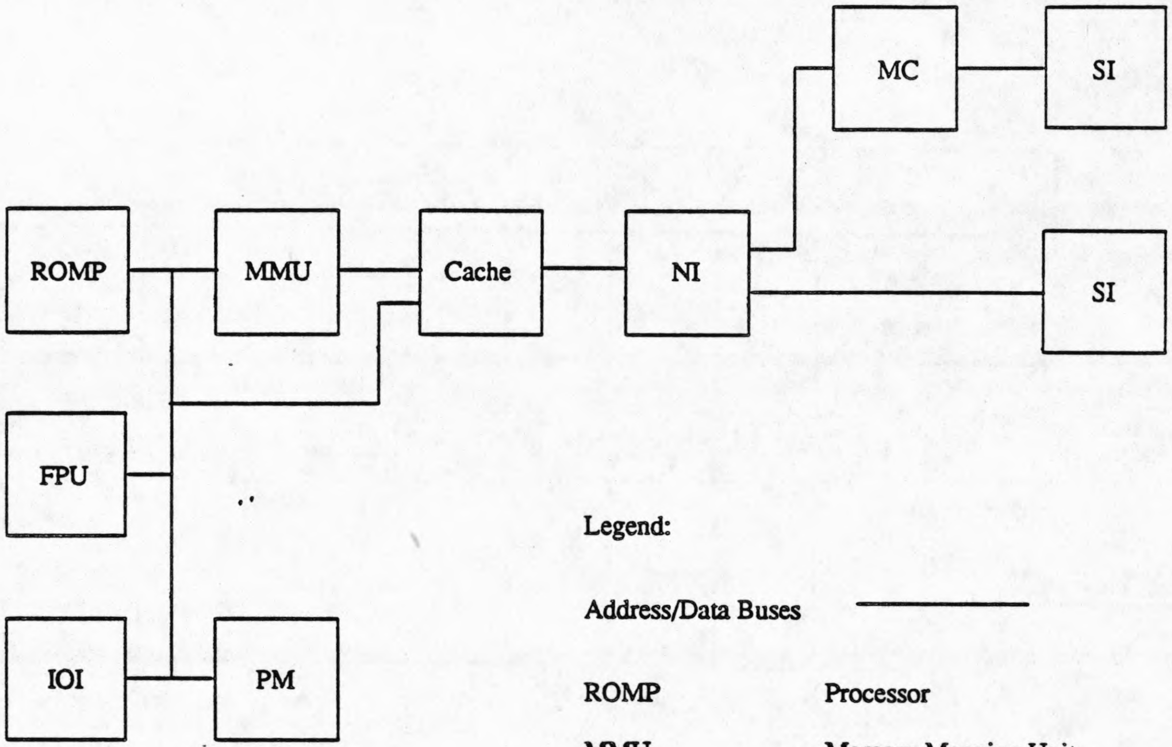


Figure 1. 64 Processor Prototype of RP3.



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. Processor-Memory Element (PME) of RP3.

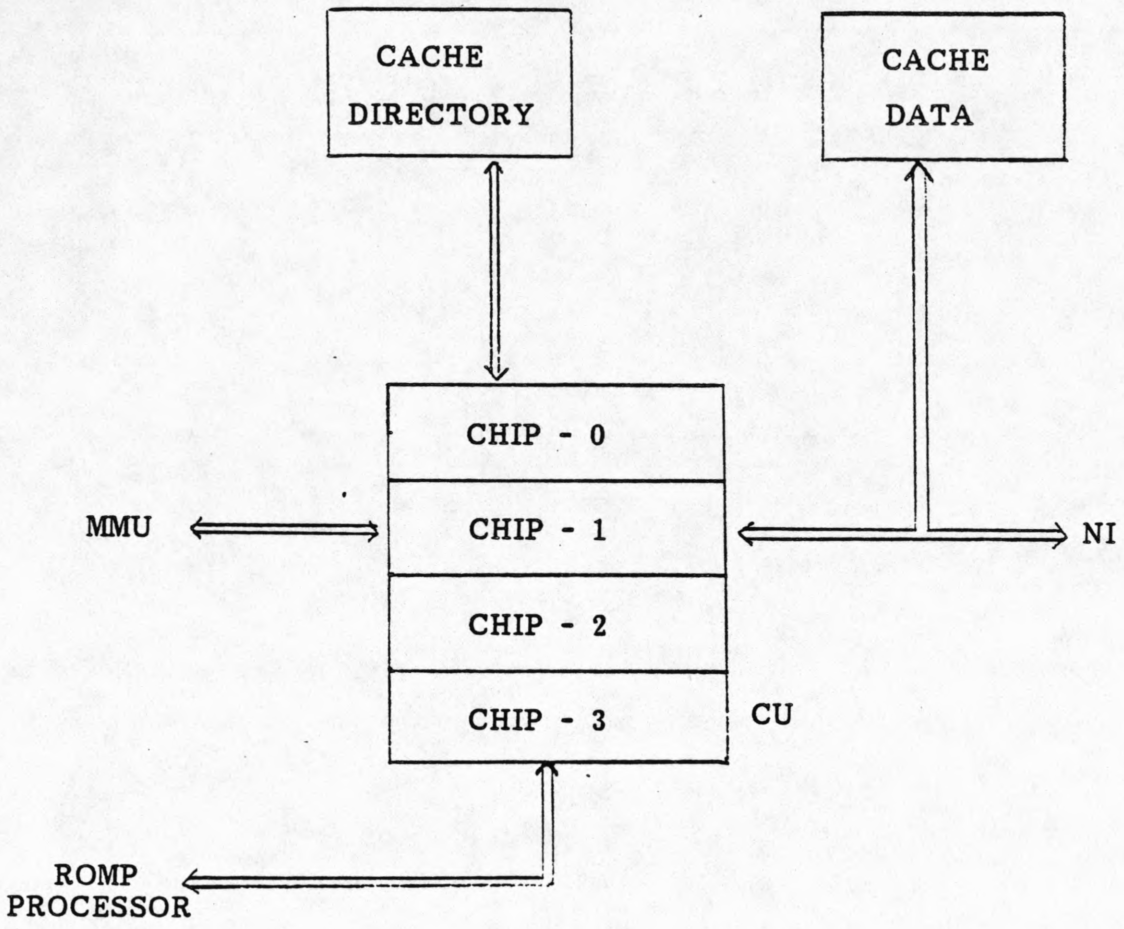


FIGURE - 3: RP3 Cache Unit

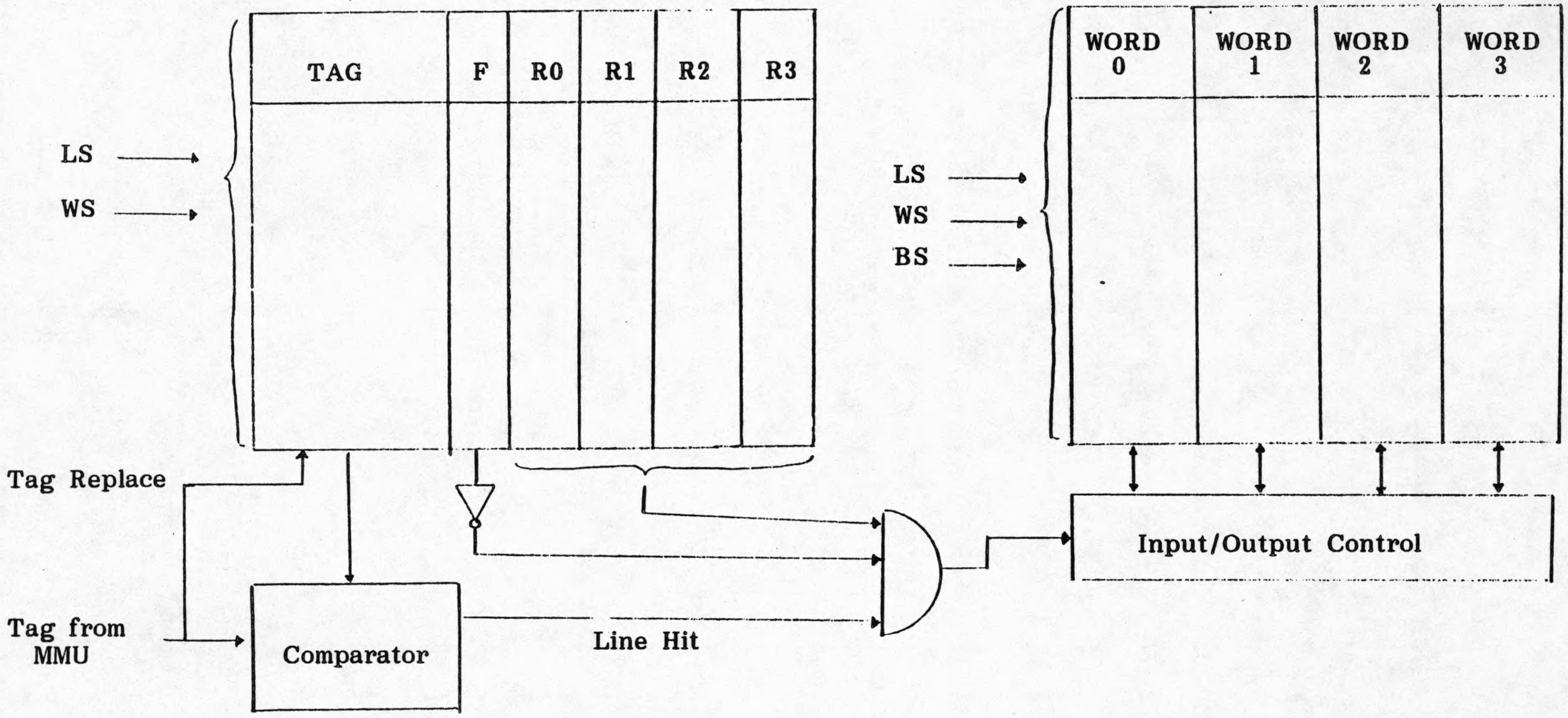
Key	Interleave	MD	V/S	F	X	R0 to R3

(a) Directory Memory

WORD 0	WORD 1	WORD 2	WORD 3

(b) Data Memory

FIGURE - 4: Cache Memory Organization



Tag = $(A_0 .. A_{17}$/Interleave/MD/V/S)

LS = $(A_{18} .. A_{27})$

WS = $(A_{28} \quad A_{29})$

BS = $(A_{30} \quad A_{31})$

FIGURE - 5: Cache Unit Organization

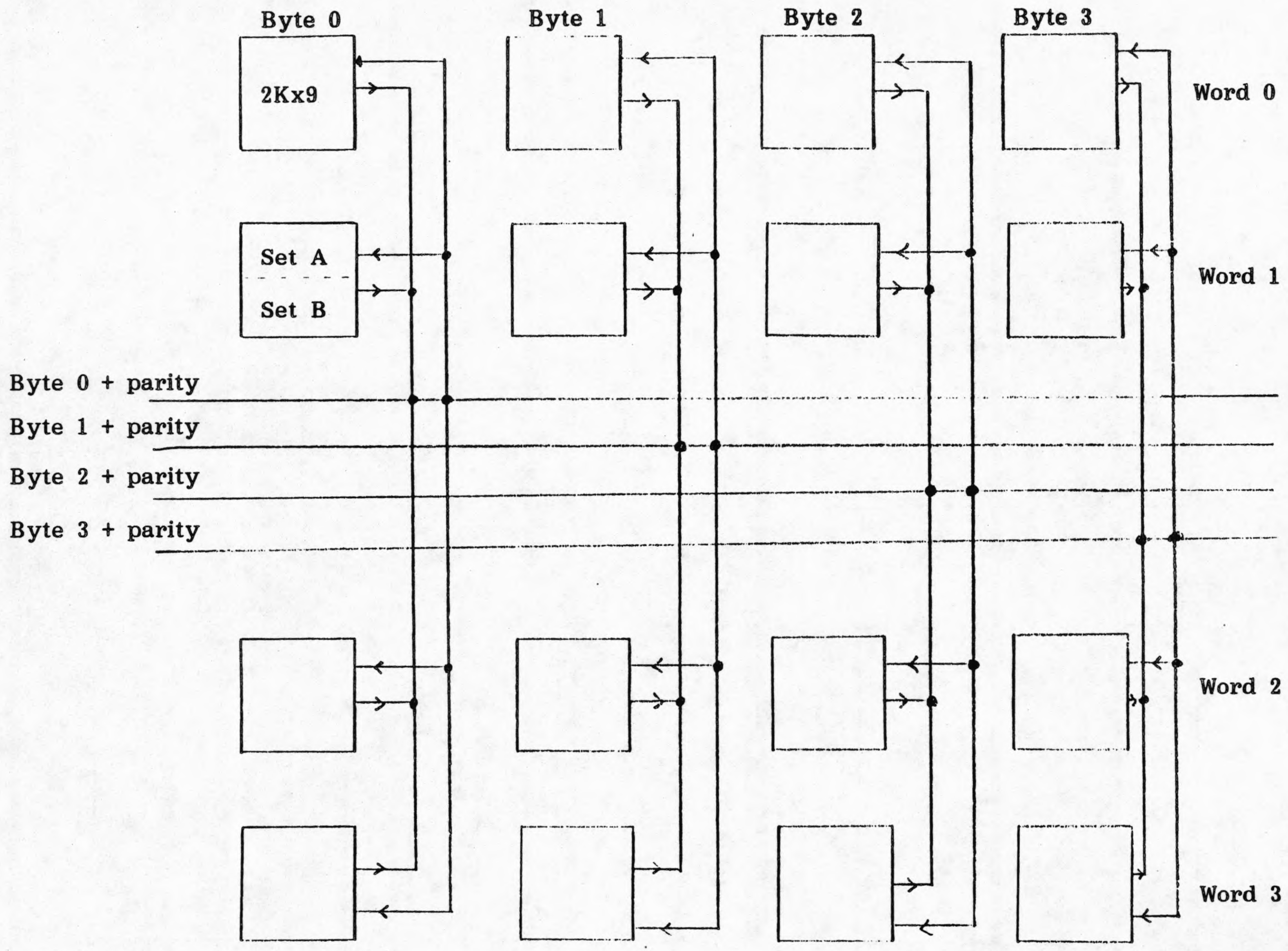


FIGURE - 6: Cache Data Memory Layout

Step / Address	1	...	n-2	n-1	n	n+1	n+2	...	3n-5	3n-4	3n-3	3n-2	3n-1	3n	3n+1	3n+2	...	5n-5	5n-4	5n-3	5n-2
A_0					Wr1								R1	Wr0							R0
A_1	Wr0					R0	Wr1								R1	Wr0					
.		.						.									.				
.		.						.									.				
A_{n-2}			Wr0						R0	Wr1								R1	Wr0		
A_{n-1}				Wr0							R0	Wr1								R1	

R0 = Read 0

R1 = Read 1

Wr0 = Write 0

Wr1 = Write 1

FIGURE - 7: MATS+ Algorithm [NAIR79]

OOOO	OOOO	
FFFF	FFFF	Data Patterns (Even Parity)
OFOF	OFOF	
FOFO	FOFO	
OO7F	7FOO	Parity Patterns (Odd Parity)
7FOO	OO7F	
3333	3333	
CCCC	CCCC	Data Patterns (Even Parity)
5555	5555	
AAAA	AAAA	

(a) Memory Test Patterns

OOOO	FFFF	Data Patterns (Even Parity)
FFFF	OOOO	
7FOO	OOOO	
OO7F	OOOO	Parity Patterns (Odd Parity)
OOOO	7FOO	
OOOO	OO7F	
OOFF	OOFF	Data Patterns (Even Parity)
FFOO	FFOO	

(b) Additional Patterns for bus test

FIGURE - 8: Test Patterns