

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

LZLABS GMBH

Petitioner

v.

INTERNATIONAL BUSINESS MACHINES CORPORATION

Patent Owner

IPR2023-00274
Patent 8,713,289

**DECLARATION OF JAMES L. MULLINS, Ph.D
PERTAINING TO PETITION FOR INTER PARTES REVIEW OF
U.S. PATENT NO. 8,713,289**

TABLE OF CONTENTS

	<u>Page</u>
I. INTRODUCTION	3
II. QUALIFICATIONS	4
III. BACKGROUND ON PUBLIC ACCESSIBILITY	6
IV. OPINION REGARDING AUTHENTICITY AND PUBLIC ACCESSIBILITY	18
V. AVAILABILITY FOR CROSS-EXAMINATION	40
VI. RIGHT TO SUPPLEMENT	40
VII. JURAT	40

I, James L. Mullins, PhD, declare as follows:

I. INTRODUCTION

1. I have been retained by petitioner LzLabs GmbH (“LzLabs”) in the above-captioned *inter partes* review relating to U.S. Patent 8,713,289 to provide opinions on various documents.

2. I am presently Dean Emeritus of Libraries and Esther Ellis Norton Professor Emeritus at Purdue University. My career as a professional and academic/research librarian/professor spanned more than 44 years including library positions at Indiana University, Villanova University, Massachusetts Institute of Technology, and Purdue University. Appendix A is a true and correct copy of my curriculum vitae describing my background and experience.

3. In 2018, I founded the firm Prior Art Documentation Librarian Services, LLC, now located at 205 Saint Cuthbert, Williamsburg, VA 23188 after purchasing the intellectual property of and successor to Prior Art Documentation, LLC located at 711 South Race Street, Urbana, IL 61801. Further information about my firm, Prior Art Documentation Librarian Services, LLC (PADLS), is available at www.priorartdoclib.com.

4. I have been retained by LzLabs to offer my opinion on the authenticity and dates of public accessibility of various documents. For this service, I am being paid my usual hourly fee of \$275.00. I have no stake in the outcome of this

proceeding or any related litigation or administrative proceedings, and my compensation in no way depends on the substance of my testimony or the outcome of this proceeding.

II. QUALIFICATIONS

5. I received a Bachelor of Arts degree in History, Religion and Political Science in 1972 as well as a Master of Arts degree in Library Science in 1973 from the University of Iowa. I received my Ph.D. in Academic Library Management in 1984 from Indiana University. Over forty-four years, I held various positions as a leader in the field of library and information sciences.

6. I am presently Dean Emeritus of Libraries and Esther Ellis Norton Professor Emeritus at Purdue University and have been since January 1, 2018. I have been previously employed as follows:

- Dean of Libraries and Professor and Esther Ellis Norton Professor, Purdue University, West Lafayette, IN (2004-2017)
- Assistant/Associate Director for Administration, Massachusetts Institute of Technology (MIT) Libraries, Cambridge, MA (2000-2004)
- University Librarian and Director, Falvey Memorial Library, Villanova University, Villanova, PA (1996-2000)
- Director of Library Services, Indiana University South Bend, South Bend, IN (1978-1996)
- Part-time Instructor, School of Library and Information Science, Indiana University, Bloomington, IN (1979-1996)
- Associate Law Librarian, and associated titles, Indiana University School of Law, Bloomington, IN (1974-1978)

- Catalog Librarian, Assistant Professor, Georgia Southern College (now University), Statesboro, GA (1973-1974)

7. I am a member of the American Library Association (“ALA”), where I served as the chair of the Research Committee of the Association of College and Research Libraries (“ACRL”). My service to ALA included service on the editorial board of the most prominent library journal, *College and Research Libraries*. I also served on the Standards Committee, College Section of the Association of College and Research Libraries, where I was instrumental in developing a re-issue of the *Standards for College Libraries* in 2000.

8. I am an author of numerous publications in the field of library science and have given presentations in library sciences at national and international conferences. During more than 44 years as an academic librarian and library science scholar, I have gained extensive experience with catalog records and online library management systems (LMS) built using Machine-Readable Cataloging (“MARC”) standards. As an academic library administrator, I have had responsibility to ensure that students were educated to identify, locate, assess, and integrate information garnered from research library resources. I have also facilitated the research of faculty colleagues either directly or through the provision of and access to the requisite print and/or digital materials and services at the universities where I worked.

9. Based on my experience identified above and detailed in my curriculum vitae, which is attached hereto as Appendix A, I consider myself to be an expert in the field of library science and academic library administration. I have previously offered my opinions on the public availability and authenticity of documents in over 100 cases. I have been deposed in one case in February 2019.

III. BACKGROUND ON PUBLIC ACCESSIBILITY

10. *Scope of This Declaration.* I am not a lawyer, and I am not rendering an opinion on the legal question of whether a particular document is, or is not, a “printed publication” under the law. I am, however, rendering my expert opinion on the authenticity of the document referenced herein and when and how this document was disseminated or otherwise made available to the extent that researchers interested and ordinarily skilled in the subject matter or art, exercising reasonable diligence, could have located the document.

11. I am informed by counsel that an item is considered authentic if there is sufficient evidence to support a finding that the item is what it is claimed to be. I am also informed that authenticity can be established based on the contents of the document itself, such as the appearance, content, substance, internal patterns, or other distinctive characteristics of the item.

12. I am informed by counsel that a given reference qualifies as “publicly accessible” if it were disseminated or otherwise made available such that a person

interested in and ordinarily skilled in the relevant subject matter could locate it through the exercise of ordinary diligence.

13. While I understand that the determination of public accessibility under the foregoing standard rests on a case-by-case analysis of the facts particular to an individual publication, I also understand that a printed publication is rendered “publicly accessible” if it is cataloged and indexed by a library such that a person interested in the relevant subject matter could locate it (*i.e.*, I understand that cataloging and indexing by a library is sufficient, though there are other ways that a printed publication may qualify as “publicly accessible”). One manner of sufficient indexing is indexing according to subject matter. I understand that it is not necessary to prove someone actually looked at the printed publication in order to show it was publicly accessible by virtue of a library’s cataloging and indexing thereof. I understand that cataloging and indexing by a single library of a single instance of a particular printed publication is sufficient. I understand that, even if access to a library is restricted, a printed publication that has been cataloged and indexed therein is publicly accessible so long as a presumption is raised that the portion of the public concerned with the relevant subject matter would know of the printed publication. I also understand that the cataloging and indexing of information that would guide a person interested in the relevant subject matter to the printed publication, such as the

cataloging and indexing of an abstract for the printed publication, is sufficient to render the printed publication publicly accessible.

14. I understand that evidence showing the specific date when a printed publication became publicly accessible is not necessary. Rather, routine business practices, such as general library cataloging and indexing practices, can be used to establish an approximate date on which a printed publication became publicly accessible.

15. *Person of Ordinary Skill in the Art.* In forming the opinions expressed in this declaration, I have reviewed the documents and appendices referenced herein. These materials are records created in the ordinary course of business by publishers, libraries, indexing services, and others. From my years of experience, I am familiar with the process for creating many of these records, and I know that these records are created by people with knowledge of the information contained within the record. Further, these records are created with the expectation that researchers and other members of the public will use them. All materials cited in this declaration and its appendices are of a type that experts in my field would reasonably rely upon and refer to in forming their opinions.

16. I have been informed by counsel that the subject matter of this proceeding relates generally to the emulation of computer system architectures and,

more particularly, to methods of emulating instructions designed for one computer architecture on another system with a different set of executable instructions.

17. I have been informed by counsel that a “person of ordinary skill in the art at the time of the inventions” (POSITA) is a hypothetical person who is presumed to be familiar with the relevant field and its literature at the time of the inventions. This hypothetical person is also a person of ordinary creativity, capable of understanding the scientific principles applicable to the pertinent field.

18. It is my opinion that such a person would have been actively engaged in academic research and learning through study and practice in the field, and possibly through formal instruction through the bibliographic resources relevant to his or her research. By the 2000s, such a person would have had access to a vast array of print resources, including at least the documents referenced below, as well as to a fast-changing set of online resources.

Library Catalog Records and Other Resources

19. Some background on MARC (Machine-Readable Cataloging) formatted records, OCLC, and WorldCat is helpful to understand the library catalog records discussed in this declaration. I am fully familiar with the library cataloging standard known as the MARC standard, which is an industry-wide standard method

of storing and organizing library catalog information.¹ MARC practices have been consistent since the MARC format was developed by the Library of Congress in the 1960s, and by the early 1970s became the U.S. national standard for disseminating bibliographic data. By the mid-1970s, MARC format became the international standard, and persists through the present. A MARC-compatible library is one that has a catalog consisting of individual MARC records for each of its items. The underlying MARC format (computer program) underpins the online public access catalog (OPAC) that is available to library users to locate a particular holding of a library. Today, MARC is the primary communications protocol for the transfer and storage of bibliographic metadata in libraries.² The MARC practices discussed

¹ The full text of the standard is available from the Library of Congress at <http://www.loc.gov/marc/bibliographic/>.

² Almost every major library in the world uses a catalog that is MARC-compatible. See, e.g., Library of Congress, *MARC Frequently Asked Questions (FAQ)*, <https://www.loc.gov/marc/faq.html> (last visited Jan. 24, 2018) (“MARC is the acronym for MACHine-Readable Cataloging. It defines a data format that emerged from a Library of Congress-led initiative that began nearly forty years ago. It provides the mechanism by which computers exchange, use, and interpret bibliographic information, and its data elements make up the foundation of most library catalogs used today.”). MARC is the ANSI/NISO Z39.2-1994 (reaffirmed 2009) standard for Information Interchange Format.

below were in place during the late 1990s time frame relevant to the documents referenced herein.

20. Online Computer Library Center (OCLC) is a not-for-profit worldwide consortium of libraries. Similar to MARC standards, OCLC's practices have been consistent since the 1970s through the present. Accordingly, the OCLC practices discussed below were in place during the time frame discussed in my opinions section. OCLC was created "to establish, maintain and operate a computerized library network and to promote the evolution of library use, of libraries themselves, and of librarianship, and to provide processes and products for the benefit of library users and libraries, including such objectives as increasing availability of library resources to individual library patrons and reducing the rate of rise of library per-unit costs, all for the fundamental public purpose of furthering ease of access to and use of the ever-expanding body of worldwide scientific, literary and educational knowledge and information."³ Among other services, OCLC and its members are responsible for maintaining the WorldCat database (<http://www.worldcat.org/>), used by libraries throughout the world.

³ OCLC Online Computer Library Center, Inc., *Amended Articles of Incorporation of OCLC Online Computer Library Center, Inc.*, Third Article (OCLC, Dublin, Ohio) Revised November 30, 2016, <https://www.oclc.org/content/dam/oclc/membership/articles-of-incorporation.pdf>.

21. Libraries worldwide use the machine-readable MARC format for catalog records. MARC-formatted records include a variety of subject access points based on the content of the document being cataloged. A MARC record for a particular work comprises several fields, each of which contains specific data about the work. Each field is identified by a standardized, unique, three-digit code corresponding to the type of data that follows. For example, a work's title is recorded in field 245, the primary author of the work is recorded in field 100, a work's International Standard Book Number ("ISBN") is recorded in field 020, and the work's Library of Congress call number (assigned by Library of Congress) is recorded in field 050. Some fields can contain subfields, which are indicated by letters. For example, a work's publication date is recorded in field 260 under the subfield "c."

22. The MARC Field 040, subfield "a," identifies the library or other entity that created the catalog record in the MARC format. MARC Field 008 identifies the date when this first MARC record was created.

23. MARC Field 008 provides the date the book was first originally cataloged, that is the first record created, typically in the United States, by the Library of Congress. The 005 date is the earliest date a book would have been available after publication.

24. MARC Field 005 indicates when the book was last ‘touched’ by the local library, typically when it was cataloged by accessing the catalog record from OCLC *WorldCat* that had been created by the Library of Congress on the date shown in MARC Field 008.

25. MARC records also include several fields that include subject matter classification information. An overview of MARC record fields is available through the Library of Congress at <http://www.loc.gov/marc/bibliographic/>. For example, 6XX fields are termed “Subject Access Fields.”⁴ Among these, for example, is the 650 field; this is the “Subject Added Entry – Topical Term” field. *See* <http://www.loc.gov/marc/bibliographic/bd650.html>. The 650 field is a “[s]ubject added entry in which the entry element is a topical term.” *Id.* The 650 field entries “are assigned to a bibliographic record to provide access according to generally accepted thesaurus-building rules (e.g., *Library of Congress Subject Headings* (LCSH), *Medical Subject Headings* (MeSH)).” *Id.* Thus, a researcher can easily discover material relevant to a topic of interest with a search using the terms employed in MARC Fields 6XX.

26. Further, MARC records include call numbers, which themselves include a classification number. For example, the 050 field is dedicated as the

⁴ *See* <http://www.loc.gov/marc/bibliographic/bd6xx.html>.

“Library of Congress Call Number”⁵ as assigned by the Library of Congress. A defined portion of the Library of Congress Call Number is the classification number, and “source of the classification number is *Library of Congress Classification* and the *LC Classification-Additions and Changes*.” *Id.* Thus, included in the 050 field is a subject matter classification. As an example: TK5105.59 indicates books on computer networks – security measures. When a local library assigns a classification number, most often a Library of Congress derived classification number created by a local library cataloger or it could be a Dewey Decimal classification number for example, 005.8, computer networks – security measures, it appears in the 090 field. In either scenario, the MARC record includes a classification number in the call number field that represents a subject matter classification.

27. The 9XX fields, which are not part of the standard MARC 21 format,⁶ were defined by OCLC for use by the Library of Congress, processing or holding notes for a local library, and for internal OCLC use.

28. For example, the 955 field is reserved for use by the Library of Congress to track the progress of a new acquisition from the time it is submitted for Cataloging in Publication (CIP) review until it is published and fully cataloged and publicly available for use within the Library of Congress. Fields 901-907, 910, and

⁵ See <http://www.loc.gov/marc/bibliographic/bd050.html>.

⁶ See <https://www.oclc.org/bibformats/en/9xx.html>.

945-949 have been defined by OCLC for local use and will pass OCLC validation. Fields 905 or 910 are often used by an individual library for internal processing purposes, for example the date of cataloging and/or the initials of the cataloger.

29. *WorldCat* is the world's largest public online catalog, maintained by the OCLC, a not-for-profit international library consortium, and built with the records created by the thousands of libraries that are members of OCLC. OCLC provides bibliographic and abstract information to the public based on MARC-compliant records through its OCLC *WorldCat* database. *WorldCat* requires no knowledge of MARC tags and code and does not require a login or password. *WorldCat* is easily accessible through the World Wide Web to all who wish to search it; there are no restrictions to be a member of a particular community, etc. The date a given catalog record was created (corresponding to the MARC Field 008) appears in some detailed *WorldCat* records as the Date of Entry but not necessarily all. *WorldCat* does not provide a view of the underlying MARC format for a specific *WorldCat* record. In order to see the underlying MARC format, the researcher must locate the book in a holding library listed among those shown in *WorldCat* and search the online public catalog (OPAC) of a holding library. Whereas *WorldCat* records are widely available, the availability of library specific MARC formatted records varies from library to library. When a specific library wishes to make the

underlying MARC format available there will be a link from the library's OPAC display, often identified as a MARC record or librarian/staff view.

30. When a MARC record is created by the Library of Congress or an OCLC member institution, the date of creation for that record is automatically populated in the fixed field (008), with characters 00 through 05 in year, month, day format (YYMMDD).⁷ Therefore, the MARC record creation date reflects the date on which the publication associated with the record was first cataloged. Thereafter, the local library's computer system may automatically update the date in field 005 every time the library updates the MARC record (*e.g.*, to reflect that an item has been moved to a different shelving location within the library, or a reload of the bibliographic data with the introduction of a new library management system that creates and manages the OPAC).

Monograph Publications

31. Monograph publications are written on a single topic, presented at length, and distinguished from an article and include books, dissertations, proceedings of a conference, and technical reports. A library typically creates a catalog record when the monograph is acquired by the library. First, it will search OCLC to determine if a record has already been created by the Library of Congress

⁷ Some of the newer library catalog systems also include hour, minute, second (HHMMSS).

or another OCLC institution. If a record is found in OCLC, the record is downloaded into the library's LMS (Library Management System) that includes typically the OPAC (online public access catalog by which researchers locate a particular library holding in a user-friendly format), acquisitions, cataloging, and circulation integrated functions. Once the item is downloaded into the library's LMS, the library adds its identifier to the OCLC database so when a search is completed on *WorldCat*, the library will be indicated as an owner of the title. Once a record is created in a Library's LMS, it is searchable and viewable through the library's OPAC, typically by author, title, and subject heading, at that library and from anywhere in the world through the internet by accessing that library's OPAC. The OPAC also connects with the circulation function of the library, which typically indicates whether the record is available, in circulation, etc., with its call number and location in a specific departmental/disciplinary library, if applicable. The OPAC not only provides immediate bibliographic access on-site, but it also facilitates the interlibrary loan process, which is when one publication is loaned from one library to another.

32. **IEEE Xplore** – The Institute of Electrical and Electronics Engineers is the world's largest organization for the advancement of technology with some 430,000 members in 160 countries. Known by its acronym IEEE, it has created a database, IEEE Xplore, that provides access to its hundreds of publications and those

of its publishing partners. This includes the content of over 170 journals, more than 1,400 conference proceedings, some 5,100 technical standards, 2,000 eBooks and 400 educational courses. In all, more than 3 million documents, dating from 1872 on, are searchable and available for purchase either through subscription or individually. Many of these records are accessible via Google Scholar.

<https://ieeexplore.ieee.org/Xplore/home.jsp>

33. **Wisconsin TechSearch (WTS)** – WTS is a set of services offered by the University of Wisconsin Libraries. WTS offers an array of article delivery and research services to any individual or organization who requests the specialized skills of WTS staff in locating and retrieving information, regardless of whether the individual is affiliated with the University of Wisconsin. <https://wts.wisc.edu/>

IV. OPINION REGARDING AUTHENTICITY AND PUBLIC ACCESSIBILITY

Document A. EX 1009: Gerry Kane. *MIPS R2000 RISC Architecture*. Prentice Hall, 1987, 1988. 1 volume, various paging. (“Kane”)

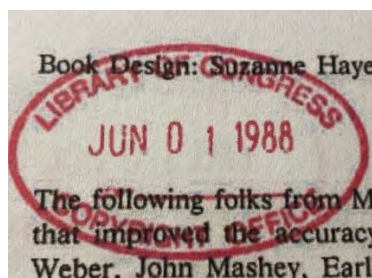
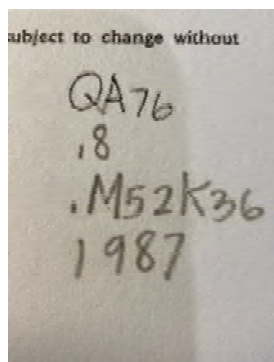
34. I have been asked to opine for a book authored by Gerry Kane titled *MIPS R2000 RISC Architecture* published by Prentice Hall [first edition] in 1987, 1988, and 1989. It is one volume with various paging containing Chapters 1-8; Appendices A-E and an Index. (“Kane”)

Authenticity

35. I have evaluated the Kane reference for authenticity in two ways: (1) by assessing a copy of Kane, Ex 1009, published in 1989, provided to me by counsel; and (2) by assessing scans of a print copy of Kane published in print in 1987, owned by the Library of Congress, that I requested through the Wisconsin TechSearch (WTS) which I received on August 10, 2022 and in a follow-up request for additional pages on October 25, 2022, Attachment A-1.

36. Attachment A-1 is a scan provided to me on August 10, 2022 and in a follow-up request on October 25, 2022 by the WTS obtained from the print copy owned by the Library of Congress. Attachment A-1 includes: a library bound front cover and spine, on the spine it reads: “Kane MIPS R2000 RISC ARCHITECTURE’ and a label that reads: “FT MEADE GenColl QA76.8 .M52 K36 1987” (as shown below) and a bar code that identifies the Library of Congress (as shown below); published front cover of Kane; title page; verso of the title page (copyright page) that has a handwritten call number that reads: “QA 76.8.M52K36 1987” and a stamp that reads “LIBRARY OF CONGRESS JUN 01 1988 COPYRIGHT OFFICE (as shown below); About this Book; Organization; Contents, vi-xiii; ; Pages: 5-18; A-1 – A-7; A-9 – A-13; A-22; A-35 – A-40; A-56 – A-58; A-

62 – A-64; A-70 – A-71; A-85; C-1 – C-2; C-8 – C-9; D-1 – D-2; and published back cover.



37. All identifying characteristics, such as stamps and notations, on Attachment A-1, are consistent with library practice and procedure that I have observed during my career as a professional librarian, specifically with those items held by the Library of Congress. I have no cause for concern about the authenticity or accuracy of these identifying attributes. In addition, Kane was found within the custody of a library, the Library of Congress, one of the most likely locations for an authentic publication to be located.

38. Evidence of the ownership of Kane by the Library of Congress will further be explored and detailed in the section below on public access.

39. After comparing Exhibit 1009 and the corresponding pages of Attachment A-1, I found no difference between the two except as noted below when discussing public accessibility. Therefore, upon finding Kane in the Library of Congress, I have determined that EX1009, Kane, is an authentic document.

40. I conclude and affirm that Kane, EX1009, is an authentic document.

Public Accessibility

41. Attachment A-2 is a screen capture of the *WorldCat* entry for Kane as copyrighted in 1987 that I completed on November 21, 2022. I obtained Attachment A-2 by completing a search on *WorldCat* by searching geographical location ‘District of Columbia (Library of Congress)’. Library of Congress was first among the 272 libraries holding Kane, 1987.

42. The copyright date, 1987, and the date of receipt, June 1, 1988, by the Library of Congress varies by one year, 1987 to 1988, due to the Library of Congress practice of assigning call numbers to the manuscript submitted to the copyright office, which more likely than not, was in 1987. The final published book was not received and cataloged/indexed by the Library of Congress until June 1, 1988 as shown by the stamp above.

43. Attachment A-2 shows that Kane is the document associated with this *WorldCat* entry, as verified by author: Gerry Kane; title: *MIPS R2000 RISC Architecture*; publisher and publication (copyright) date; Prentice-Hall in 1987: and ISBN: 0-13-5844749-4.

44. Kane, 1987, could have been located by searching for the author: Gerry Kane; title; *MIPS R2000 RISC Architecture*: and/or by searching the subject heading: *C-1-PROCESSOR ARCHITECTURES*.

45. Attachment A-3 is a screen capture of the *WorldCat* entry for Kane, copyrighted in 1989, that I completed on November 21, 2022. I obtained Attachment A-3 by completing a search on *WorldCat* on November 21, 2022 by searching geographical location, District of Columbia (Library of Congress). Library of Congress was first among the 190 libraries holding Kane, 1989.

46. Attachment A-3 is the OCLC *WorldCat* record for Kane as copyrighted in 1989, Exhibit 1009, as provided to me by counsel. Attachment A-3 shows that Kane, 1989, is the document associated with this *WorldCat* entry, as verified by author: Gerry Kane; title: *MIPS R2000 RISC Architecture*; publisher and publication (copyright) date; Prentice-Hall in 1989: and ISBN: 0-13-584293-X; 0-13-5844749-4.

47. Kane, 1989, could have been located by searching for the author: Gerry Kane; title; *MIPS R2000 RISC Architecture*; and/or by searching the subject headings: *Computer architecture*; and/or *MIPS R2000 series microprocessors*.

48. The searches discussed above could have been performed anywhere in the world by anyone who accessed *WorldCat* and its predecessor database, *FirstSearch*, through an OCLC member library in the 1990s through 2005.

49. As discussed above, after a comparison between all pages included in Exhibit 1009 with those in Attachment A-1, I found no difference in the text or content of Kane except on the title page where a paragraph has been inserted on Kane 1989, as shown below, an “Acknowledgements” section on the verso of the title page (copyright page) in Kane 1987 rather than a separate page as in Kane 1989, and revisions to text on the “About this Book” page between Kane 1987 and Kane 1989, all of which would be consistent with its release as a new printing, but not a new edition which would indicate changes in the text or content of Kane.

*This book describes the MIPS RISC Architecture.
The R2000 Processor and R2010 Floating Point Accelerator
are the original implementation of the architecture and
the R2000/R2010 numbering notation is used throughout the text.
Higher-performance versions of the two machines,
the R3000 and R3010, are the newest implementation of
the MIPS RISC architecture. All references in this book
to the R2000/R2010 Processors apply fully
to the new R3000/R3010 Processors.*

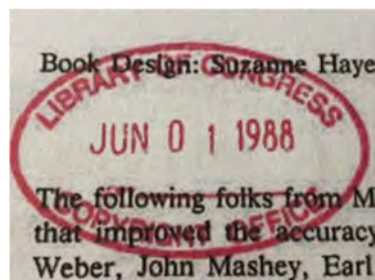
50. Attachment A-4 is a download I made from the Library of Congress OPAC (online catalog) on August 10, 2022. The document cataloged in this record

is Kane as verified by the OPAC record fields including author: Gerry Kane; main title: *MIPS R2000 RISC Architecture*; and publisher and publication date; Prentice Hall in 1987; and ISBN 01358447494.

51. I compared the Library of Congress Classification (call number): QA76.8.M52 K36 1987 with that shown on the bound cover and the verso of the title page (copyright) of Attachment A-1, it is the same on bound cover and on the verso of the title page (copyright).

52. Kane could have been located in the Library of Congress OPAC by searching for author: Gerry Kane: title: *MIPS R2000 RISC Architecture*; and/or by searching the subject headings: *MIPS R2000 series microprocessors; Computer architecture; and/or Reduced instruction set computers*.

53. The date stamped on Attachment A-1, June 1, 1988 (as shown below), indicates receipt of Kane by the Library of Congress, after which it would have been sent to the cataloging department for verification of the “CIP” information cataloging and the final descriptive cataloging completed and accessible through Library of Congress OPAC.



54. Once Kane was entered into the general collection of the Library of Congress, members of the public could access the book by having it brought to either the Jefferson or Adams Reading Rooms. The collections of the Library of Congress are searchable by subject matter, author, or title such that a skilled researcher could find works in which they were interested. For example, a member of the public could have located a copy of Kane by searching for the author: Gerry Kane; by title: *MIPS R2000 RISC Architecture*; and/or by searching the subject headings: *MIPS R2000 series microprocessors*; *Computer architecture*; and/or *Reduced instruction set computers*.

55. Members of the public could read, study, and make notes about a selected work in the Reading Rooms. Further, members of the public were permitted to make photocopies of portions of the work while in the Reading Rooms. Accordingly, a copy of Kane was accessible to the general public when it was available at the Library of Congress.

56. Attachment A-5 is the MARC record I downloaded from the Library of Congress OPAC on August 10, 2022. The MARC format provides information about the processing of Kane by the Library of Congress. As mentioned above, the 9XX field in the MARC format is allocated to local libraries to enter information specific to that library. The Library of Congress has reserved the 955 field to

indicate date of Cataloging in Publication (CIP) information, receipt of the final published book and cataloging.

57. The MARC 955 field in this record reads –

955 — |aby 13 to bt00 06-15-88; fg09 06-28-88

58. The 955 MARC field record indicates the processing of Kane for cataloging from June 15, 1988 through June 28, 1988 when full cataloging was completed.

59. The physical copy of Kane at the Library of Congress would have been available for public access within one week to ten days after it was cataloged on June 28, 1988, labelled and transferred to the stacks. Therefore, Kane would have been accessible to the public in the Library of Congress no later than July 9, 1988.

Conclusion

60. I conclude that Kane, EX1009, is an authentic document and would have been publicly accessible through the Library of Congress, more likely than not, no later than July 9, 1988.

Document B. EX 1014: Harvey G. Cragon. *Computer Architecture and Implementation*. Cambridge University Press, 2000. 317 pages. (“Cragon”)

61. I have been asked to opine a book authored by Harvey G. Cragon titled: *Computer Architecture and Implementation*. Cambridge University Press, 2000. 317 pages. (“Cragon”)

Authenticity

62. I have evaluated the Cragon reference for authenticity in two ways: (1) by assessing the digital copy of Cragon, Ex 1014, provided to me by counsel; and (2) by assessing scans of a print copy of Cragon owned by the Library of Congress, that I requested through the Wisconsin TechSearch (WTS) that I received October 31, 2022, Attachment B-1.

63. Attachment B-1 is a scan provided to me at my request on October 31, 2022 by the WTS obtained from the print copy owned by the Library of Congress. Attachment B-1 includes: front cover, back cover, and spine, on the cover a label reads: “QA 76.9.A73 C74 2000 Copy 2 Ft Meade GenColl”(as shown below); title page; verso of the title page (copyright page) that has a handwritten call number that reads: “QA 76.9.A73C74 2000 Copy 2” and a stamp that reads “LIBRARY OF CONGRESS March 28 2000 COPYRIGHT OFFICE (as shown below); Contents; Preface; and Pages: 57, 88, 89, 90, 91, 130, 276, 277, 278, 279, 300, 301, 308, 309.

QA 76
.9
.A73
C74
2000
COPY 2

FT MEADE
GenColl

QA76
.9
.A73C74
2000
Copy 2



64. All identifying characteristics, such as stamps and notations, on Attachment B-1, are consistent with library practice and procedure that I have observed during my career as a professional librarian, specifically with those items held by the Library of Congress. I have no cause for concern about the authenticity or accuracy of these identifying attributes. In addition, Cragon was found within the custody of a library, the Library of Congress, one of the most likely locations for an authentic publication to be located.

65. Evidence of the ownership of Cragon by the Library of Congress will further be explored and detailed in the section below on public access.

66. After comparing Exhibit 1014 and the corresponding pages of Attachment B-1, I found no difference between the two. Therefore, upon finding Cragon in the Library of Congress I have determined that EX1014, Cragon, is an authentic document.

67. I conclude and affirm that Cragon, EX1014, is an authentic document.

Public Accessibility

68. Attachment B-2 is a true and correct copy of the *WorldCat* entry for Cragon. I obtained Attachment B-2 by completing a search on *WorldCat* on October 25, 2022 by searching District of Columbia (Library of Congress).

69. Attachment B-2 shows that Cragon is the document associated with this *WorldCat* entry, as verified by author: Harvey G. Cragon; title: *Computer Architecture and Implementation*. publisher and publication date Cambridge University Press in 2000: and ISBN: 0-521-65168-9 (Hardback).

70. Cragon could have been located by searching for the author: Harvey G. Cragon; title: *Computer Architecture and Implementation* and/or by searching the subject headings: *Architecture (Computers)*; *Computer Design*; *Computers – Computer Architecture*; and/or *Input/Output Routines*.

71. When I searched *WorldCat* for holdings of Cragon in the District of Columbia, the Library of Congress was on the list among the 310 libraries shown as holding Cragon worldwide.

72. The search discussed above could have been performed anywhere in the world by anyone who accessed *WorldCat* and its predecessor database, *FirstSearch*, through an OCLC member library in the 1990s through 2005.

73. Attachment B-3 is a download I made from the Library of Congress OPAC (online catalog) on October 25, 2022. The document cataloged in this record is Cragon as verified by the OPAC record fields including author: Harvey G. Cragon; main title: *Computer Architecture and Implementation* and/or by searching the subject headings: *Architecture (Computers)*; *Computer Design*; *Computers – Computer Architecture*; and/or *Input/Output Routines.*; and ISBN 0-521-65168-9 (Hardback).

74. I compared the Library of Congress Classification (call number): QA76.9.A73 C74 2000 (with the exception of Copy 2) with that shown on the bound cover and the verso of the title page (copyright) of Attachment B-1, it is the same on bound cover and on the verso of the title page (copyright)..

75. Cragon could have been located in the Library of Congress OPAC by searching for author: Harvey G. Cragon: title: *Computer Architecture and Implementation*; and/or by searching the subject heading: *Computer architecture*.

76. The date stamped on Attachment B-1, March 28, 2000 (as shown below), indicates receipt of Cragon by the Library of Congress, after which it would have been sent to the cataloging department for verification of the “CIP” information cataloging and the final descriptive cataloging completed and accessible through Library of Congress OPAC.



77. Once Cragon was entered into the general collection of the Library of Congress, members of the public could access the book by having it brought to either the Jefferson or Adams Reading Rooms. The collections of the Library of Congress are searchable by subject matter, author, or title such that a skilled researcher could find works in which they were interested. For example, a member of the public could have located a copy of Cragon by searching for the author: Harvey G. Cragon; by title: *Computer Architecture and Implementation*; and/or by searching the subject heading: *Computer architecture*.

78. Members of the public could read, study, and make notes about a selected work in the Reading Rooms. Further, members of the public were permitted to make photocopies of portions of the work while in the Reading Rooms. Accordingly, a copy of Cragon was accessible to the general public when it was available at the Library of Congress.

79. Attachment B-4 is the MARC record I downloaded from the Library of Congress OPAC on October 25, 2022. The MARC format provides information about the processing of Cragon by the Library of Congress. As mentioned above,

the 9XX field in the MARC format is allocated to local libraries to enter information specific to that library. The Library of Congress has reserved the 955 field to indicate date of Cataloging in Publication (CIP) information, receipt of the final published book and cataloging.

80. The MARC 955 field in this record reads –

955 |apc03 to ja00 05-25-99; jf03 to Subj. 05-25-99; jf11 to sl 05-26-99; jf12 05-26-99 to ddc;
CIP ver. jg15 05-03-00; ja15 to BCCD 07-13-00

81. The 955 MARC field record indicates the processing of Cragon for CIP cataloging of the manuscript from May 25, 1999 through May 26, 1999. Once the published book was received by the Library of Congress (March 28, 2000) it was sent through cataloging to verify CIP preliminary cataloging and final cataloging including pagination, etc., that began on May 3, 2000 and was completed on July 13, 2000.

82. The physical copy of Cragon at the Library of Congress would have been available for public access within one week to ten days after it was cataloged on July 13, 2000, labelled and transferred to the stacks. Therefore, Cragon would have been accessible to the public in the Library of Congress no later than July 23, 2000.

Conclusion

83. I conclude that Cragon, EX1014, is an authentic document and would have been publicly accessible through the Library of Congress, more likely than not, no later than July 23, 2000.

Document C: EX. 1012. H. Umeno, M. Kiyama, T. Fukunaga and T. Kubo, "New Method for Dispatching Waiting Logical Processors in Virtual Machine System," *Proceedings, 29th Annual International Computer Software and Applications Conference (COMPSAC'05)*, July 26-28, 2005, Volume 2, pp. 551-560. ("Umeno")

Authentication

84. I have been asked to opine on an article authored by H. Umeno, et al., titled "New Method for Dispatching Waiting Logical Processors in Virtual Machine System," published in the proceedings of the *29th Annual International Computer Software and Applications Conference (COMPSAC'05)*, held July 26-28, 2005, in volume 2, on pages 551-560. ("Umeno").

85. For consistency and clarification, Umeno, when used in this declaration, refers not only to the Umeno article but also the *Proceedings, 29th Annual International Computer Software and Applications Conference (COMPSAC'05)*, July 26-28, 2005 in which it was published.

86. I have evaluated the Umeno reference three ways: (1) by assessing the digital copy of Umeno, EX 1012, provided to me by counsel; (2) by assessing scans of a print copy of Umeno owned by the North Carolina State University Libraries

provided to me at my request on November 7, 2022. Attachment C-1; and (3) accessing and reviewing Umeno in digital format available to me through Purdue University Libraries, Attachment C-2.

87. Attachment C-1 was provided to me at my request on November 7, 2022, by North Carolina State University Libraries. Attachment C-1 includes scans of Umeno: cover; inside front cover with a stamp that reads “North Carolina State University – The Libraries”, as shown below, title page of *Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC 2005)*, July 26-28, 2005, Edinburgh, Scotland, sponsored by IEEE Computer Society, Supported by Motorola; verso of the title page (copyright); Contents, pages v-xii; and the Umeno article on pages 551-560.



88. All identifying characteristics, such as stamps and notations on Attachment C-1 are consistent with library practice and procedure that I have observed during my career as a professional librarian. I have no cause for concern about the authenticity or accuracy of these identifying attributes. Attachment C-1 was found within the custody of a library, the North Carolina State University

Libraries, one of the most likely locations for an authentic document to be publicly accessible. Evidence of the ownership of Umeno by the North Carolina State University Libraries will further be explored and detailed in the section below on public access.

89. Attachment C-2 is a download of Umeno from the *IEEE Xplore* database made by me on November 6, 2022 enabled by my emeritus faculty status at Purdue University through Purdue University Libraries at this URL:

<https://ieeexplore-ieee-org.ezproxy.lib.purdue.edu/stamp/stamp.jsp?tp=&arnumber=1510083>

It is available for a fee at this URL:

<https://ieeexplore.ieee.org/abstract/document/1510083>

90. After comparing EX 1012, Umeno, with Attachment C-1 and the corresponding pages of Attachment C-2, I found no difference between EX 1012, Umeno, and Attachment C-1 and Attachment C-2. Therefore, accessing and retrieving Umeno from a database through the Purdue University Libraries and upon finding Umeno in the North Carolina State University Libraries, I have determined that EX 1012, Umeno, is an authentic document.

91. I conclude and affirm that Umeno, EX 1012, is an authentic document.

Public Accessibility

92. Attachment C-3 is a true and correct copy of the *WorldCat* entry for Umeno. I obtained Attachment C-3 by completing a search on *WorldCat* on November 8, 2022 by searching the geographical location, North Carolina, for the North Carolina State University Library.

93. Attachment C-3 shows that Umeno is the document associated with this *WorldCat* entry, as verified by title: *Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC 2005)*, July 26-28, 2005, Edinburgh, Scotland.

94. Umeno could have been located in *WorldCat* by searching for the title: *Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC 2005)*, July 26-28, 2005, Edinburgh, Scotland, and/or by searching the subject headings: *Computer networks*; *Computer network – Congresses*; *Computer programming*; *Computer programming – Congresses*; and/or *Programming languages (Electronic computers)*.

95. When I searched *WorldCat* for holdings of Umeno in North Carolina, the North Carolina State University Library was first on the list among the 19 libraries shown as holding Umeno worldwide.

96. The search discussed above could have been performed anywhere in the world by anyone who accessed *WorldCat* in 2005 and thereafter.

97. Attachment C-4 is a download I made from the North Carolina State University Library OPAC (online catalog) on November 6, 2022. The document cataloged in this record is Umeno as verified by the OPAC record fields including title: *Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC 2005)*, July 26-28, 2005, Edinburgh, Scotland.; call number: QA76.6.C6295 2005 v.2 and ISBN: 0-7695-2413-3.

98. Umeno could have been located in the North Carolina State University Library OPAC by searching for the title: *Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC 2005)*, July 26-28, 2005, Edinburgh, Scotland, and/or by searching the subject headings: *Computer network – Congresses; Computer programming – Congresses; and/or Programming languages (Electronic computers) – Congresses.*

99. Although there is an ownership stamp on the inside front cover page, as shown below of the North Carolina State University Library, it does not have a date of acquisition/cataloging/indexing. Therefore, it is necessary to turn to the OPAC MARC record to document when Umeno was acquired/cataloged/indexed.



005 20051012031732.0

008 050829t20052005caua b 101 0 eng d

100. Attachment C-5 is the OPAC MARC record for Umeno. Inserted above are the 005 and 008 MARC Fields for Umeno. As mentioned above in the introduction, the 008 MARC Field gives the date a book was originally cataloged most typically by the Library of Congress. The 008 field indicates Umeno was cataloged on August 29, 2005 (050829...remaining digits are check digits).

101. The MARC Field 005 indicates when the local library, in this case the North Carolina State University Library, last ‘touched’ the record for the item. Touched in this instance was when Umeno was cataloged by the North Carolina State University Library, which is October 12, 2005 (20051012...remaining digits are check digits).

102. In order to verify that the 005 MARC Field did reflect the cataloging date of Umeno by the North Carolina State University Library, I asked North Carolina State University Library staff to verify that I was reading the date correctly and that it would reflect, more likely than not, the date of cataloging by the North Carolina State University Library staff.

103. Attachment C-6 includes several emails sent by me to the staff of the North Carolina State University Library staff and their response. Inserted below is a quote from an email from Beth Ashmore, Associate Head, Acquisitions and Discovery (Serials), North Carolina State University Libraries sent to me on

November 7, 2022. In this email she corroborates that using the dates in the 005 MARC Field is an appropriate substitute for a date accession stamp. She is quoted below from Attachment C-6:

Hi Mia and Dr. Mullins,

I agree that the 005 is an appropriate substitute for a dated accession stamp. 10/12/2005 matches the cataloged date on our bibliographic control screen in our catalog. Unfortunately, we don't have any order record information in our system that would tell us more specifically when the item was received prior to cataloging.

Lynn, if there is another place to check for this info, let me know.

Hope that helps!

-Beth

Beth Ashmore

Associate Head , Acquisitions and Discovery (Serials)

North Carolina State University Libraries

beashmor@ncsu.edu

919-513-3354

104. The physical copy of Umeno at the North Carolina State University Library would have been available for public access within one week to ten days after it was cataloged/indexed on October 12, 2005. Therefore, Umeno would have been accessible to the public in the North Carolina State University Library no later than October 22, 2005.

105. I conclude and affirm that Umeno, EX 1012, was publicly accessible by at least October 22, 2005.

Conclusion

106. I conclude that Umeno, EX 1012 is an authentic document and would have been publicly accessible through the North Carolina State University Library no later than October 22, 2005.

V. AVAILABILITY FOR CROSS-EXAMINATION

107. In signing this Declaration, I recognize that this Declaration will be filed as evidence in a contested case before the Patent Trial and Appeal Board of the U.S. Patent and Trademark Office. I also recognize that I may be subject to cross-examination in the case and that cross-examination will take place within the United States. If cross-examination is required of me, I will appear for cross-examination within the United States during the time allotted for cross-examination.

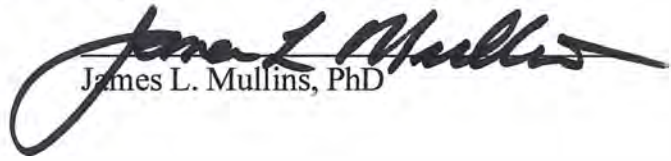
VI. RIGHT TO SUPPLEMENT

108. I reserve the right to supplement my opinions in the future to respond to any arguments that the Patent Owner raises and to take into account new information as it becomes available to me.

VII. JURAT

I declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the full knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the U.S. Code

Dated: November 27, 2022


James L. Mullins, PhD

JAMES L. MULLINS, PhD

Prior Art Documentation Librarian Services, LLC

205 Saint Cuthbert, Williamsburg, VA 23188

jlmullins@priorartdoclib.com

ph. 765 479 4956

Prior Art Documentation Librarian Services, LLC. (PADLS). Founded January 2018. As of August, 2022, 107 declarations have been completed for 36 law firms; four depositions were scheduled, three cancelled a few days prior, and only one was held in February, 2019.

Library Experience:

2018-present Dean Emeritus, Libraries & Norton Professor Emeritus, Purdue University
2011 - 2017 Dean of Libraries & Esther Ellis Norton Professor, Purdue University
2004 - 2011 Dean of Libraries & Professor, Purdue University, West Lafayette, IN
**2000-2004 Assistant/Associate Director for Administration, MIT Libraries,
Massachusetts Institute of Technology, Cambridge, MA.**
**1996-2000 University Librarian & Director, Falvey Memorial Library. Villanova University,
Villanova, PA.**
1978-1996 Director of Library Services, Indiana University South Bend.
**1974-1978 Associate Librarian, Indiana University Bloomington, School of Law. 1973-
1974 Instructor/Catalog Librarian. Georgia Southern College (now University).**

Teaching Experience:

**1977-1996 Associate Professor (part-time), School of Library and Information Science, Indiana
University. Subjects taught: Cataloging, Management, and Academic Librarianship.**

Education:

1972 - University of Iowa. Honors Bachelor of Arts, History, Religion & Political Science.
1973 - University of Iowa. Master of Arts in Library Science.
**1984 - Indiana University. Doctor of Philosophy. Concentration: Academic Library Administration.
Emphasis: Law Librarianship.**

Awards and Recognition:

2017 Wilmeth Active Learning Center/Library of Engineering and Science, Grand Reading Room, was announced by President Mitch Daniels, Purdue University, that it would be re-named the James L. Mullins Reading Room to honor his leadership and reputation in the academic library profession.

September 2017. Portrait unveiled December 2017.

2017 Distinguished Alumnus Award by the School of Informatics and Computing, Indiana University, Bloomington. Given June 25, 2017.

2016 Hugh C. Atkinson Memorial Award, jointly sponsored by the four divisions of the American Library Association (ALA), June 27, 2016.

2015 ACRL Excellence in University Libraries Award, April 23, 2015.

Named Esther Ellis Norton Professor of Library Science by Purdue Trustees, December 11, 2011.

International Review Panel to evaluate the University of Pretoria Library, February 20 – 24, 2011. Pretoria, South Africa.

Publications: (selected)

Designing the Academic Library, Chapter Contributor. Association of College and Research Libraries, American Library Association, 2022.

A Purdue Icon: Creation, Life, and Legacy, edited by James L. Mullins, Founder's Series, Purdue University Press, 138pp., August 2017.

"The policy and institutional framework." In *Research Data Management, Practical Strategies for Information Professionals*, edited by Ray, J M. Purdue University Press, pp.25-44, 2014.

"DataCite: linking research to data sets and content." In Benson, P and Silver, S. *What Editors Want: An Author's Guide to Scientific Journal Publishing*. University of Chicago Press, pp. 21-23, December 2012.

"Library Publishing Services: Strategies for Success," with R. Crow, O. Ivins, A. Mower, C. Murray-Rust, J. Ogburn, D Nesdill, M. Newton, J. Speer, C. Watkinson. *Scholarly Publishing and Academic Resources Coalition (SPARC)*, version 2.0, March 2012.

"The Changing Definition and Role of Collections and Services in the University Research Library." *Indiana Libraries*, Vol 31, Number 1 (2012), pp.18-24.

"Are MLS Graduates Being Prepared for the Changing and Emerging Roles that Librarians must now assume within Research Libraries?" *Journal of Library Administration*. Volume 52, Issue 1, 2012, p. 124-132

Baykoucheva, Svetla. What Do Libraries Have to Do with e-Science?: An Interview with James L. Mullins, Dean of Purdue University Libraries. *Chem. Inf. Bull.* [Online] 2011, 63 (1), 45-49.

<http://www.acscinf.org/publications/bulletin/63-1/mullins.php> (accessed Mar 16, 2011).

“The Challenges of e-Science Data-set Management and Scholarly Communication for Domain Sciences and Technology: a Role for Academic Libraries and Librarians,” chapter in, *The Digital Deluge: Can Libraries Cope with e-Science?*” Deanna B. Marcum and Gerald George, editors, Libraries Unlimited/Teacher Ideas Press, 2009. (a monograph publication of the combined proceedings of the KIT/CLIR proceedings).

“Bringing Librarianship to e-Science,” *College and Research Libraries*. vol. 70, no. 3, May 2009, editorial.

“The Librarian’s Role in e-Science” *Joho Kanri (Journal on Information Processing and Management)*, Japan Science and Technology Agency (formerly Japan Information Center of Science and Technology), Tokyo, Japan. Translated into Japanese by Taeko Kato. March 2008.

The Challenge of e-Science Data-set Management to Domain Sciences and Engineering: a Role for Academic Libraries and Librarians, ” KIT (Kanazawa Institute of Technology)/CLIR (Council of Library and Information Resources) International Roundtable for Library and Information Science, July 5-6, 2007. Developments in e-science status quo and the challenge, The Japan Foundation, 2007.

“An Administrative Perspective,” Chapter 14, *Proven Strategies for Building an Information Literacy Program*, Susan Curzon and Lynn Lampert, editors, Neal-Schuman Publishers, Inc., New York, 2007. pp. 229-237.

Library Management and Marketing in a Multicultural World, proceedings of the IFLA Management and Marketing (M&M) Section, Shanghai, China, August 16-17, 2006, edited. K.G. Saur, Munchen, Germany, June 2007. 390 pp.

Top Ten Assumptions for the Future of Academic Libraries and Librarians: a report from the ACRL Research Committee, with Frank R. Allen and Jon R. Hufford. *College & Research Libraries*, April 2007, vol.68, no.4. pp.240-241, 246.

To Stand the Test of Time: Long-term Stewardship of Digital Data Sets in Science and Engineering. A report to the National Science Foundation from the ARL Workshop on New Collaborative Relationships: the Role of Academic Libraries in the Digital Data Universe. September 26-27, 2006, Arlington, VA. p.141. <http://www.arl.org/bm~doc/digdatarpt.pdf>

“Enabling Interaction and Quality in a Distributed Data DRIS,” *Enabling Interaction and Quality: Beyond the Hanseatic League*. 8th International Conference on Current Research Information Systems, with D. Scott Brandt and Michael Witt. Promoted by euro CRIS. Leuven University Press, 2006. pp.55-62. Editors: Anne Garns Steine Asserson and Eduard J. Simons.

"Standards for College Libraries, the final version approved January 2000," prepared by the ACRL College Libraries Standards Committee (member), *C&RL News*, March 2000, p.175-182.

"Standards for College Libraries: a draft," prepared by the ACRL College Libraries Section, Standards Committee (member), *C&RL News*, May 1999, p. 375-381.

"Statistical Measures of Usage of Web-based Resources," *The Serials Librarian*, vol. 36, no. 1-2 (1999) p. 207-10.

"An Opportunity: Cooperation between the Library and Computer Services," in *Building Partnerships: Computing and Library Professionals*. Edited by Anne G. Lipow and Sheila D. Creth. Berkeley and San Carlos, CA, Library Solutions Press, 1995. p. 69-70.

"Faculty Status of Librarians: A Comparative Study of Two Universities in the United Kingdom and How They Compare to the Association of College and Research Libraries Standards, " in *Academic Librarianship, Past, Present, and Future: a Festschrift in Honor of David Kaser*. Englewood, Colorado; Libraries Unlimited, 1989. p. 67-78. Review in: *College & Research Libraries*, vol. 51, no. 6. November 1990, p. 573-574.

Presentations: (Representative)

“How Long the Odyssey? Transitioning the Library and Librarians to Meet the Needs and Expectations of the 21st Century University,” David Kaser Lecture, School of Informatics & Computing, Indiana University, Bloomington, IN, November 16, 2015.

Presentation at University of Cape Town, Cape Town, South Africa, August 20, 2015.

“The Challenge of Discovering Science and Technology Information,” Moderator, International Federation of Library Associations (IFLA) Science and Technological Libraries Section Program, Cape Town, South Africa, August 18, 2015.

“An Odyssey in Data Management: Purdue University,” International Federation of Library Associations (IFLA) Research Data Management: Finding Our Role – A program of the Research Data Alliance, Cape Town, South Africa, August 17, 2015.

Presentation at University of Pretoria, Pretoria, South Africa, August 11, 2015.

Co-Convener with Sarah Thomas, Harvard University, at the Harvard Purdue Symposium on Data Management, Harvard University, Cambridge, MA, June 15-18, 2015.

“Strategic Communication,” panel discussion on the Director's role and perspective on library communications at Committee on Institutional Cooperation (CIC) Center for Library Initiatives (CLI) Annual Conference, University of Illinois Urbana-Champaign, May 20, 2015.

“Issues in Data Management,” panel discussion moderated by Catherine Woteki, United States Undersecretary for Research, Education & Economics at 20th Agriculture Network Information Collaborative (AgNIC) Annual Meeting in the National Agricultural Library, Beltsville, MD, May 6, 2015.

“Active learning/IMPACT & the Active Learning Center at Purdue University,” Florida Institute of Technology, Melbourne, FL, February 11, 2015.

“Science+art=creativity: libraries and the new collaborative thinking,” panel moderator, International Federation of Library Associations (IFLA) 80th General Conference and Assembly, Lyon, France, August 19, 2014.

“Purdue University The Active Learning Center—A new concept for a library,” Association of University Architects 59th Annual National Conference, University of Notre Dame, South Bend, IN, June 23, 2014.

“Big Data & Implications for Academic Libraries,” keynote speaker, Greater Western Library Alliance (GWLA) Cyber-infrastructure Conference, Kansas City, MO, May 28, 2014.

“Research Infrastructure,” panel moderator, Association of Research Libraries (ARL) 164th Membership Meeting, Ohio State University, Columbus, OH, May 7, 2014.

“An Eight Year Odyssey in Data Management: Purdue University,” International Association of Scientific and Technological University Libraries (IATUL) 2013 Workshop Research Data Management: Finding Our Role, University of Oxford, UK, December 2013.

“Purdue University Libraries & Press: from collaboration to integration,” Ithaka Sustainable Scholarship, The Evolving Digital Landscape: New Roles and Responsibilities in Higher Education, libraries as publishers, New York, New York, October 2013.

“Tsinghua and Purdue: Research Libraries for the 21st Century,” Tsinghua University, Tsinghua, China, August 2013.

“Purdue Publishing Experience in the Libraries Publishing Coalition,” Association of American University Presses Annual Meeting, Press-Library Coalition Panel, Boston, Massachusetts, June 21, 2013.

“Indiana University Librarians Day: Purdue University Libraries Ready for the 21st Century,” Indiana University Purdue University Indianapolis (IUPUI), June 7, 2013.

“Purdue University Libraries and Open Access; CNI Project Update,” Coalition for Networked Information, San Antonio, TX, April 5, 2013.

Memorial Resolution, honoring Joseph Brannon, to the Board of the Association of College & Research Libraries, Seattle, WA, January 2013.

“An overview of sustaining e-Science collaboration in an Academic Research Library—the Purdue experience,” Duraspace e-Science Institute webcast, October 17, 2012.

“The Role of Libraries in Data Curation, Access, and Preservation: an International Perspective,” Panel Moderator, 78th General Conference and Assembly, International Federation of Library Associations, Helsinki, Finland, August 15, 2012.

“21st Century Libraries,” moderator of First Plenary Session, International Association of Technological University Libraries 33rd Annual Conference, Singapore, June 4, 2012.

“Planning for New Buildings on Campus,” panel presenter, University of Calgary Building Symposium on Designing Libraries for the 21st Century, Calgary, Alberta, Canada, May 17, 2012.

“Data Management and e-Science, the Purdue Response.” Wiley-Blackwell Executive Seminar-2012, Washington, DC, March 23, 2012.

“An overview of Sustaining e-Science Collaboration in Academic Research Libraries and the Purdue Experience.” Leadership & Career Development Program Institute, Association of Research Libraries (ARL). Houston, TX, March 21, 2012.

“An overview of Data Activities at Purdue University in response to Data Management Requirements.” Coalition for Academic Scientific Computation (CASC). Arlington, VA, September 8, 2011.

“Getting on Track with Tenure,” Association of College and Research Libraries (ACRL) Research Program Committee. Washington, DC, June 26, 2011.

“Integration of the Press and Libraries Collaboration to Promote Scholarly Communication,” Association of Library Collections & Technical Services (ALCTS) Scholarly Communication Interest Group – American Library Association, New Orleans, Louisiana, June 25, 2011.

“Cooperation for improving access to scholarly communication,” with N. Lossau (Germany), C. Mazurek (Poland), J. Stokker (Australia), panel moderator and presenter, Second Plenary Session, International Association of Scientific and Technological University Libraries (IATUL) 32nd Conference 2011, Warsaw, Poland. May 29-June 2, 2011.

“Riding the Wave of Data,” STM Annual Spring Conference 2011. Trailblazing & transforming scholarly publishing 2011. Washington, D.C., April 28, 2011.

“Confronting old assumptions to assume new roles: physical and operational integration of the Press and Libraries at Purdue University,” keynote speaker, 2011 BioOne Publishers & Partners Meeting. Washington, D. C., April 22, 2011.

“Are MLS Graduates Being Prepared for the Changing and Emerging Roles that Librarians must now assume within Research Libraries?” University of Oklahoma Libraries Seminar, March 4, 2011, Oklahoma City, Oklahoma.

“The Future Role of University Librarians,” the University of Cape Town, South Africa, February 25, 2011.

“New Roles for Librarians: the Application of Library Science to Scientific/Technical Research – Purdue University – a case study. International Council for Science and Technology (ICSTI); Ottawa, Canada. June 9, 2009.

“Reinventing Science Librarianship: Models for the Future,” Association of Research Libraries / Coalition for Networked Information. October 16-17th, 2008, Arlington, VA. Moderator and convener of Data Curation: Issues and Challenges.

“Practical Implementation and Opportunities Created at Purdue University,” African Digital Curation Conference, Pretoria, South Africa, (live video transmission), February 12, 2008.

Keynote speaker. “*Scholarly Communication & Academe: The Winter of Our Discontent*,” XXVII Charleston Conference on Issues in Book and Serial Acquisition, Charleston, South Carolina. November 8, 2007.

Keynote speaker. “*Enabling Access to Scientific & Technical Data-sets in e-Science: a role for Library and Archival Sciences*,” Greater Western Library Alliance (GWLA), Tucson, Arizona. September 17, 2007. A meeting of library directors and vice presidents for research of member institutions.

“*The Challenge of e-Science Data-set Management to Domain Sciences and Engineering: a Role for Academic Libraries and Librarians*,” KIT (Kanazawa Institute of Technology)/CLIR (Council of

Library and Information Resources) International Roundtable for Library and Information Science, July 5-6, 2007. Invited to participate by the Deputy Librarian of Congress.

International Association of Technological University Libraries (IATUL), Stockholm, Sweden. June 8, 2007. Invited paper, *Enabling International Access to Scientific Data-sets: creation of the Distributed Data Curation Center (D2C2)*.

“A New Collaboration for Librarians: The Principles of Library and Archival Sciences Applied to the Curation of Datasets,” Symposium of the Libraries and the College of Engineering, University of Louisville, April 6, 2007.

“Purdue University Libraries: Through Pre-eminent Innovation and Creativity, Meeting the Challenges of the Information Age,” Board of Trustees, Purdue University, February 15, 2007.

ARL Workshop on New Collaborative Relationships: The Role of Academic Libraries in the Digital Data Universe, September 26-27, 2006, Arlington, VA. Invited participant.

NARA and SDSC: A partnership. A panel before the National Science Foundation, June 27, 2006. Arlington, VA. Invited participant.

“Kaleidoscope of Scientific Literacy: fusing new connections,” with Diane Rein, American Library Association, Association of College and Research Libraries, Science & Technology Section, Annual Conference, New Orleans, June 26th, 2006.

“Leadership for Learning: Building a Culture of Teaching in Academic Libraries – an administrative perspective,” American Library Association, Association of College and Research Libraries, Instruction Section, Annual Conference, New Orleans, June 25th, 2006.

“Building an interdisciplinary research program in an academic library:

how the Libraries’ associate dean for research makes a difference at Purdue University,” International Association of Technological University Libraries (IATUL), Porto, Portugal, May 23rd, 2006.

“Enabling Interaction and Quality in a Distributed Data DRIS,” *Enabling Interaction and Quality: Beyond the Hanseatic League*. 8th International Conference on Current Research Information Systems, with D. Scott Brandt and Michael Witt. Promoted by euro CRIS, Bergen, Norway, May 12th, 2006, Brandt, and Witt presented in person

“Interdisciplinary Research,” with D. Scott Brandt, Coalition for Networked Information (CNI) Spring Meeting: Project Briefing, Washington, D.C., April 3rd, 2006.

“An Interview with Purdue’s James Mullins,” a podcast submitted by Matt Pasiewicz, on *Educause Connect*, http://connect.educause.edu/James_L_Mullins_Interview_CNI_2005

“Managing Long-Lived Digital Data-sets and their Curation: Interdisciplinary Policy Issues,” Managing Digital Assets Forum, Association of Research Libraries (ARL), Washington, D.C., October 28th, 2005.

“The Odyssey of a Librarian.” Indiana Library Federation (ILF), District 2 Meeting, South Bend, Indiana. October 4th, 2005.

"New College Library Standards," Standards Committee Presentation, ALA, Chicago, July 7, 2000.
SUNY Library Directors, Lake George, New York. “*The College Library Standards: a Tool for Assessment.*” April 5, 2000.

Tri-State College Library Association, *Finding You Have Talents You Never Knew You Had*, Penn State Great Valley, March 25, 2000.

Using Web Statistics, American Library Association, New Orleans, June 24, 1999.

Keynote speaker at the JSTOR Workshop, January 29, 30, 1999. University of Pennsylvania, Philadelphia, PA.

"The New Standards for Electronic Resources Statistics," Society of Scholarly Publishers, Washington, D.C., September 17, 1998.

"Evaluating Online Resources: Now that you've got them what do you do?," joint presenter with Chuck Hamaker, LSU, at the NASIG Conference, Boulder, Colorado. June 1998.

"What Employers Are Looking for in New Librarians?" Pennsylvania Library Association, Philadelphia. September 26, 1997.

"The Theory of Matrix Management" panel presentation of the Comparative Library Organization Committee of the Library Organization and Management Section of the Library Administration and Management Association, a division of the American Library Association, Annual Meeting, Chicago, June 24, 1990.

Professional Involvement: (summary of recent emphasis)

The focus for my professional involvement and research has moved recently toward managing massive data-sets. This has resulted in working with faculty in the sciences and technology to determine how librarians can collaborate in managing, curating, and preserving data-sets for future access and documentation. This has included various speaking opportunities as well as participation in planning with the National Science Foundation (NSF) on ways in which librarians can be integrated more completely into the funded research process. Participation in the Kanazawa Institute of Technology/Council of Library Resources Roundtable was particularly rewarding and provided new opportunities to share with international colleagues the issues surrounding data-set management. I was

the champion for the creation of the Distributed Data Curation Center (D2C2) at Purdue University (<http://d2c2.lib.purdue.edu/>)

Throughout my career, beginning with my dissertation, I have been actively involved with assessing and evaluating libraries. In the fall of 1999, I contacted twenty-two academic library directors to determine whether the need was also felt by others. The response was overwhelmingly affirmative. This resulted in a meeting at ALA Midwinter, January 2000. A formal meeting followed at Villanova University in April 2000. As convener, I helped to form the University Libraries Group (ULG), modeled after the Oberlin Group for college libraries. The ULG is made up of university libraries that support diverse wide-ranging programs through doctoral level and have a level of support that places them in the top tier of academic institutions. A few of the member libraries, along with Villanova, are William and Mary, Wake Forest, Lehigh, Carnegie-Mellon, Tufts, Marquette, Miami of Ohio, and Southern Methodist. In 1994 appointed to the Standards Committee, College Section, Association of College and Research Libraries. During the next six years, the Committee concentrated on changing the focus of the standards from quantitative analysis of input and output factors to emphasis on assessment of the outcome. Culmination of the work was a re-issue of the *Standards for College Libraries* in 2000. The knowledge gained through my work experience enabled me to formulate the changes needed in the standards. This work allowed for close collaboration with accrediting agencies, both professional and regional. During this same time another focus emerged, the impact of digital resources. Through my work on the JSTOR Statistics Task Force, standards were developed on the collection of use of electronic databases. This Standard was later adopted in 1998 by the International Consortium of Library Consortia (ICOLC). In 2002, the American Library Association appointed me to serve as the liaison to the Marketing and Management Section of the International Federation of Library Associations (IFLA).

Professional Service: (representative list)

Nominations Committee, Association of Research Libraries (ARL), 2016.

Steering Committee, Scholarly Publishing and Academic Resources Coalition (SPARC), 2016 – 2017.

“Excellence in Library Services,” Chair, Review Team, University of Hong Kong, Hong Kong, August 24-27, 2015.

Chair, Management Advisory Board, 2015-2017; Member, Scientific Advisory Board, arXiv, Cornell University, 1/1/2013 – present.

Advisory Board for the Wayne State University School of Library and Information Science, July 2012 – present.

Advisory Board for Microsoft Academic Search, 2012 – 2015. Redmond, WA.

Transforming Research Libraries, a Strategic Direction Steering Committee of the Association of Research Libraries (ARL), 2012-2015.

Science and Technology section, representing ARL, International Federation of Library Associations (IFLA), Chair, 2013 – 2017; Member, 2011 to present.

Member of University of Pretoria, South Africa, Library Review Committee. August 2013.

Co-chair, Local Arrangements Planning Committee for 2013 Conference, Association of College and Research Libraries (ACRL), a division of the American Library Association (ALA).

Association of Research Libraries Leadership & Career Development Program Mentor, 2011-2017.

e-Science Task Force, Association of Research Libraries. July 2006 – present. Chair, October 2011 – October 2012.

Board of Directors, International Association of Technological University Libraries (IATUL). January 2008 – December 2014.

Midwest Collaborative for Library Services (MCLS); Board Member, October 2010 – December 2012.

Chair, Library Directors, Committee on Institutional Cooperation (CIC), July 2010 – June 2012.

Board of Directors, Association of Research Libraries (ARL); October 2008 – October 2011.

Scholarly Communication Steering Committee, Association of Research Libraries (ARL) 2008-2011.

Editorial Board, *College and Research Libraries*, Association of College and Research Libraries, American Library Association. January 2008 – December 2014.

Chair, Organizing Committee for IATUL Conference 2010, June 21-24, 2010, Purdue University, West Lafayette, Indiana/Chicago, Illinois.

Conference Planning Committee for National Conference of the Association of College and Research Libraries, 2009, Seattle, Washington.

Research Committee, Association of College and Research Libraries, ACRL, division of ALA. 2002-2007, chair, 2005-2007.

Association of Research Libraries, Search and Screen Committee, Executive Director. March – January 2008.

Center for Research Libraries, Board of Directors. April 2006 – April 2012.

Academic Libraries of Indiana, Board of Directors, 2004 – present. Vice-president, 2005-2007. President, 2007- 2009.

ALA Representative to the International Federation of Library Associations (IFLA), Marketing and Management (M&M) Section, initial term 2003-2007, re-appointed for second term, 2007-2011.

Invited to represent Research Libraries at the ACRL/3M Wonewok Retreat to assess Marketing of Academic Libraries, October 2002.

Hugh A. Atkinson Award Committee, LAMA Representative, ALA, 2001-2005.

Program Committee, Library Administrators and Management Association (LAMA), a division of ALA. 1996-2001.

ACRL, Standards and Accreditation Committee, a division of ALA. Liaison to RBMS Section of ACRL. 1997-2002.

Elected to the Executive Committee of LAMA, LOMS, a division of the American Library Association, 1998-2000. Nominated as Chair/Elect for 2003 – 2005.

Columbia University Press Advisory Committee. 1996 - 2000.

LITA/LAMA Conference Evaluation Committee, Pittsburgh, Pennsylvania, October 1996.

"New Learning Communities," Coalition for Networked Information, Indianapolis. November 19-21, 1995. Facilitator for invitational, national conference committed to developing collaborative learning and teaching techniques, involving librarians.

Planning Committee-Evaluation. LITA/LAMA 1996 Conference, Pittsburgh. This first conference, to be held jointly between two divisions of ALA, will focus on new technologies within libraries.

Indiana Cooperative Library Services Authority (InCoLSA), elected to Executive Committee, April 1991, served as President in 1993-94. InCoLSA is a statewide network of academic, public, school, and special libraries that supports library cooperation for cataloging, interlibrary loan, collection development and application of new technologies.

Governor's Conference on Libraries and Information Services. Served on Planning Committee, Academic Libraries Representative, appointed by the Governor to represent academic libraries in Indiana, Chair, Finance Committee, April 1989-July 1991.

Indiana Library Endowment Foundation Board, 1984-92. Charter Member, 1984, President, 1988-1992. 2004-2005.

University Service: (Summary)

Served on search and screen committees for senior positions including chancellor, dean, and directors; most recently I have been asked to serve on the search committee for the provost of Purdue University. At MIT service included the Library Council & appointment to the Administrative Council by President

Vest, 2001-2003 & Member of the Faculty Committee on the Library System. At Purdue appointed by the President to the Search Committee for the Provost, October 2007 to May 2008; member of the Capital Projects Committee, and IT Operational Oversight Committee as senior academic dean, 2008-2014.

Global Council, Global Policy Institute, 2012 – 2016.

Academic Program Excellence and Rankings (APER) project team, 2014.

Representative of the Academic Deans on the Re-engineering Business Operations, Purdue University, 2016 –

Academic Deans Council chaired by Provost – 2004 – 2017.

University Promotion and Tenure Committee – 2006 – 2017.

"Outstanding Team Award, Electronic Reserve Project," served as Chair, recognition awarded by the President of Villanova University to one team who made an outstanding contribution to the operations of the University, selected by a committee of administrators, faculty, and staff. Awarded September 9, 1999.

Nominated for the IUSB Lundquist Award, 1995 & 1996. The Lundquist award is given to faculty who have "exhibited excellence in teaching, scholarly or artistic achievement, and diversified relevant service..."

KANE

MIPS
R2000
RISC
ARCHI-
TECTURE

FT MEADE
GenColl

QA 76

.8

.M52 K36

1987

LIBRARY OF CONGRESS



00003841078

mips
R2000
RISC
ARCHITECTURE



GERRY KANE

MIPS R2000 RISC Architecture

Gerry Kane
#1



Prentice Hall , Englewood Cliffs, NJ 07632

© 1987. MIPS Computer Systems, Inc. All Rights Reserved. No part of this document may be reproduced in any form or by any means without the prior express written consent of MIPS Computer Systems, Inc. and the publisher.

The information contained in this book is for reference only and is subject to change without notice.

UNIX is a Trademark of AT&T.

Ethernet is a Trademark of XEROX.

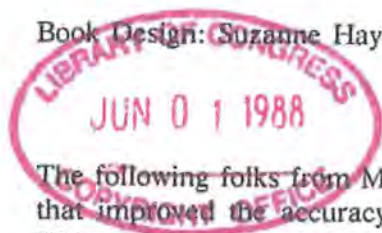
VAX is a Trademark of Digital Equipment Corp.

IBM is a Trademark of International Business Machines

68000 is a Trademark of Motorola Semiconductor Corp.

QA76
.8
.M52K36
1987

Book Design: Suzanne Hayes-Kane.



Acknowledgements

The following folks from MIPS Computer Systems contributed valuable suggestions that improved the accuracy and readability of this book: John Moussouris, Larry Weber, John Mashey, Earl Killian, Edward (Skip) Stritter. Special credit goes to Craig Hansen who wrote the original architectural specifications that served as the source material for this book.

Personal thanks go to the usual suspects: Suzanne, Sean, Kyle, Ambrose, Marcella.

This book is dedicated to the memory of RedBob (Robert A. Knox), a source of joy and energy to all who knew him. Perhaps not the soul of the MIPS machine — but certainly, the wit. He is greatly missed.

The publisher offers discounts on this book when ordered in bulk quantities. For more information, write:

Special Sales/College Marketing
Prentice-Hall, Inc.
College Technical and Reference Division
Englewood Cliffs, NJ 07632

All rights reserved. No part of this book may be reproduced, in any form or by any means, without permission in writing from the publisher.

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

ISBN 0-13-584747-4

Prentice-Hall International (UK) Limited, London
Prentice-Hall of Australia Pty. Limited, Sydney
Prentice-Hall Canada Inc., Toronto
Prentice-Hall Hispanoamericana, S.A., Mexico
Prentice-Hall of India Private Limited, New Delhi
Prentice-Hall of Japan, Inc., Tokyo
Simon & Schuster Asia Pte. Ltd., Singapore
Editora Prentice-Hall do Brasil, Ltda., Rio de Janeiro

10-60-90

About This Book

This book is the primary architectural reference manual for the R2000 family of processors. It describes the functional characteristics and capabilities of the R2000 Processor and the R2010 Floating Point Accelerator (FPA). These processors are available from the following manufacturers:

- Integrated Device Technology, Inc.
3226 Scott Boulevard
P.O. Box Box 58015
Santa Clara, CA 95054-3029
(408) 727-6116
Telex 887766
FAX (408) 988-3029
- LSI Logic Corporation
1551 McCarthy Blvd.
MS D-105
Milpitas, CA 95035
(408) 433-8000
Telex 171.641
FAX (408) 433-3816
Attn: Standard Products Marketing
- MIPS Computer Systems, Inc.
930 Arques Avenue
Sunnyvale, CA 94086-3650
(408) 720-1700
Telex 510 601 5346 -
FAX (408) 720-9809
- Performance Semiconductor Corporation
610 E. Weddell Drive
Sunnyvale, CA 94089
(408) 734-9000
FAX (408) 734-0258
Attn: Microprocessor Marketing

Organization

This book is organized into two major sections: Chapters 1 through 5 describe the characteristics of the R2000 Processor and Chapters 6, 7, and 8 describe the R2010 FPA. The contents of each chapter are summarized in the list that follows:

- **Chapter 1, RISC Architecture: An Overview**, describes the general characteristics and concepts of reduced instruction set computers.
- **Chapter 2, R2000 Processor Overview**, describes the general characteristics and capabilities of the processor. It also provides a programming model which describes how data is represented in the R2000 registers and in memory and also provides a summary of the R2000 CPU registers.
- **Chapter 3, Instruction Set Summary**, provides a summary description of the R2000 instruction set.
- **Chapter 4, Memory Management System**, describes the virtual memory system supported by the R2000's System Control Coprocessor.
- **Chapter 5, Exception Processing**, describes the events that cause R2000 exceptions and the sequences that occur during processing of the exceptions.
- **Chapter 6, R2010 FPA Overview**, describes the general characteristics and capabilities of the FPA. This chapter also provides a summary of the R2010 FPA registers and describes how data is represented in the R2010 registers.
- **Chapter 7, FPA Instruction Summary & Instruction Pipeline**, provides a summary description of the R2010 instruction set and a discussion of instruction overlapping.
- **Chapter 8, Floating Point Exceptions**, describes how the R2010 FPA supports the IEEE standard floating point exceptions.
- **Appendix A** provides a detailed description of the format and operation of each R2000 instruction.
- **Appendix B** provides a detailed description of the format and operation of each R2010 FPA instruction.

- **Appendix C** describes machine language programming tips that can simplify implementation of commonly required tasks.
- **Appendix D** describes assembly language programming techniques and provides guidelines for writing programs for use with the MIPS assembler.
- **Appendix E** describes how the R2010 FPA supports the IEEE floating point standard and provides programming tips that can simplify implementation of standard operations not implemented by the FPA.

Contents

1

RISC Architecture: An Overview

Scope of this Overview	1-1
What Is RISC?	1-1
Defining Performance	1-2
Time/Instruction	1-3
Cycles/Instruction (C)	1-4
Instruction Pipelines	1-4
Load/Store Architecture	1-7
Delayed Load Instructions	1-7
Delayed Branch Instructions	1-9
Time/Cycle (T)	1-11
Instruction Decode Time	1-11
Instruction Operation Time	1-11
Instruction Access Time	1-12
Overall Architectural Simplicity	1-13
Instructions/Task (I)	1-14
Optimizing Compilers	1-14
Operating System Support	1-17
The RISC Design Process	1-18
Hidden Benefits of RISC Design	1-19
Shorter Design Cycle	1-19
Smaller Chip Size	1-19
User (Programmer) Benefits	1-19
Most Aggressive Technologies	1-19

2

R2000 Processor Overview

R2000 Processor Features	2-2
R2000 CPU Registers	2-2
Instruction Set Overview	2-3
R2000 Processor Programming Model	2-6
Data Formats and Addressing	2-6
R2000 CPU General Registers	2-8
R2000 System Control Coprocessor	2-9
System Control Coprocessor Registers	2-9
Memory Management System	2-10
The TLB (Translation Lookaside Buffer)	2-10
R2000 Operating Modes	2-10
R2000 Pipeline Architecture	2-12
Memory System Hierarchy	2-13

3

R2000 Instruction Set Summary

Instruction Formats	3-1
Instruction Notational Conventions	3-2
Load and Store Instructions	3-2
Computational Instructions	3-5
ALU Immediate Instructions	3-5
3-Operand Register Type Instructions	3-6
Shift Instructions	3-7
Multiply/Divide Instructions	3-8
Jump and Branch Instructions	3-9
Special Instructions	3-10
Coprocessor Instructions	3-11
System Control Coprocessor Instructions	3-12
R2000 Instruction Pipeline	3-13
The Delayed Instruction Slot	3-14

Delayed Loads	3-15
Delayed Jumps and Branches	3-16

4

Memory Management System

Memory System Architecture	4-1
Privilege States	4-2
User-Mode Virtual Addressing	4-3
Kernel-Mode Virtual Addressing	4-3
Virtual Memory and the TLB	4-4
TLB Entries	4-5
EntryHi & EntryLo Registers	4-6
Virtual Address Translation	4-7
The Index Register	4-9
The Random Register	4-10
TLB Instructions	4-10

5

Exception Processing

The Exception Handling Registers	5-3
The Cause Register	5-4
The EPC Register	5-5
The Status Register	5-6
Status Register Mode Bits and Exception Processing	5-9
BadVAddr Register	5-10
Context Register	5-10
Processor Revision ID Register	5-11
Exception Description Details	5-12
Exception Vector Locations	5-12
Address Error Exception	5-13

Breakpoint Exception	5-14
Bus Error Exception	5-15
Coprocessor Unusable Exception	5-16
Interrupt Exception	5-17
Overflow Exception	5-18
Reserved Instruction Exception	5-19
Reset Exception	5-20
System Call Exception	5-21
TLB Miss Exceptions	5-22
TLB Miss Exception	5-23
TLB Mod Exception	5-25
UTLB Miss Exception	5-26

6

R2010 FPA Overview

R2010 FPA Features	6-1
R2010 FPA Programming Model	6-2
Floating-Point General Registers (FGR)	6-3
Floating-Point Registers (FPR)	6-4
Floating-Point Control Registers (FGR)	6-4
Control/Status Register	6-5
Implementation and Revision Register	6-8
Floating-Point Formats	6-8
Number Definitions	6-10
Normalized Numbers	6-10
Denormalized Numbers	6-11
Infinity	6-11
Zero	6-11
Coprocessor Operation	6-11
Load, Store, and Move Operations	6-11
Floating-Point Operations	6-12
Exceptions	6-12
Instruction Set Overview	6-13
R2010 Pipeline Architecture	6-14

7

FPA Instruction Set Summary & Instruction Pipeline

Instruction Set Summary	7-1
Load, Store, and Move Instructions	7-1
Floating Point Computational Instructions	7-3
Floating Point Relational Operations	7-5
Branch on FPA Condition Instructions	7-7
The Instruction Pipeline	7-8
Instruction Execution Times	7-10
Overlapping FPA Instructions	7-12

8

Floating Point Exceptions

Exception Trap Processing	8-2
Inexact Exception (I)	8-4
Invalid Operation Exception (V)	8-5
Division-by-Zero Exception (Z)	8-6
Overflow Exception (O)	8-7
Underflow Exception (U)	8-8
Unimplemented Operation Exception (E)	8-9
Saving and Restoring State	8-10

A

R2000 Instruction Set Details

B

R2010 FPA Instruction Set Details

C

Machine Language Programming Tips

32-bit Addresses or Constants	C-2
Indexed Addressing	C-3
Subroutine Return using JR Instruction	C-4
Jumping to 32-bit Addresses	C-4
Branching On Arithmetic Comparisons	C-4
Filling the Branch Delay Slot	C-5
Testing for Carry	C-6
Testing for Overflow	C-8
Multi-precision Math	C-10
Double-Word Shifts	C-12

D

Assembly Language Programming

Register Use and Linkage	D-1
General Registers	D-1
Special Registers	D-3
Floating Point Registers	D-3
Assembly Instruction Summaries	D-4
Addressing	D-9
Address Formats	D-10
Address Descriptions	D-11
Pseudo Op-Codes	D-12
Linkage Conventions	D-18
Examples	D-24
Memory Allocation	D-27
Basic Machine Definition	D-28
Load and Store Instructions	D-28
Computational Instructions	D-28
Branch Instructions	D-29
Coprocessor Instructions	D-29
Special Instructions	D-29

E

IEEE Floating Point Standard Compatibility Issues

Interpretation of the Standard	E-1
Underflow	E-1
Exceptions	E-1
Inexact	E-2
NaNs	E-2
Software Assistance for IEEE Compatibility	E-2
IEEE Exception Trapping	E-3
IEEE Format Compatibility	E-3
Implementing IEEE Standard	
Operations in Software	E-4
Remainder	E-4
Round to Integer	E-4
Convert between Binary and Decimal	E-5
Copy Sign	E-5
Scale Binary	E-5
Log Binary	E-5
Next After	E-5
Finite	E-5
Is NaN	E-6
Arithmetic Inequality	E-6
Class	E-6

Index

Overflow Exception

Cause. This exception occurs when an **ADD ADDI, SUB, or SUBI** instruction results in two's complement overflow. This exception is not maskable.

Handling. The R2000 branches to the General Exception vector (0x80000080) for this exception. The R2000 sets the *OV* code of the *Cause* register.

The *EPC* register points at the instruction that caused the exception, unless the instruction is in a branch delay slot: in that case, the *EPC* register points at the **branch** instruction that preceded the exception-causing instruction and sets the *BD* bit of the *Cause* register.

The R2000 saves the *KUp*, *IEp*, *KUc*, and *IEc* bits of the *Status* register in the *KUo*, *IEo*, *KUp*, and *IEp* bits, respectively, and clears the *KUc* and *IEc* bits.

Servicing. A kernel should hand the executing process a floating point exception or integer overflow error when this exception occurs. Such an error is usually fatal.

A

R2000 Processor Instruction Set Details

This appendix provides a detailed description of the operation of each R2000 instruction. The instructions are listed in alphabetical order.

Refer to **Appendix B** for a detailed description of the R2010 FPA instructions.

The exceptions that may occur due to the execution of each instruction are listed after the description of each instruction. The description of the immediate causes and manner of handling exceptions is omitted from the instruction descriptions in this chapter. Refer to **Chapter 5** for detailed descriptions of exceptions and handling.

A chart on the last page of this appendix lists the bit encoding for the constant fields of each instruction.

Instruction Classes

R2000 instructions are divided into the following classes:

- **Load/Store** instructions move data between memory and general registers. They are all I-type instructions, since the only addressing mode supported is *base register + 16-bit immediate offset*.
- **Computational** instructions perform arithmetic, logical and shift operations on values in registers. They occur in both R-type (both operands are registers) and I-type (one operand is a 16-bit immediate) formats.
- **Jump and Branch** instructions change the control flow of a program. Jumps are always to absolute 26-bit word addresses (J-type format), or 32-bit register addresses (R-type). Branches have 16-bit offsets relative to the program counter (I-type). Jump and Link instructions save a return address in Register 31.
- **Coprocessor** instructions perform operations in the coprocessors. Coprocessor Loads and Stores are I-type. Coprocessor computational instructions have coprocessor-dependent formats (see the FPA instructions). Coprocessor zero (CP0) instructions manipulate the memory management and exception handling facilities of the processor.
- **Special** instructions perform a variety of tasks, including movement of data between special and general registers, syscall, and breakpoint. They are always R-type.

Instruction Formats

Every R2000 instruction consists of a single word (32 bits) aligned on a word boundary and there are only three instruction formats as shown in Figure A.1.

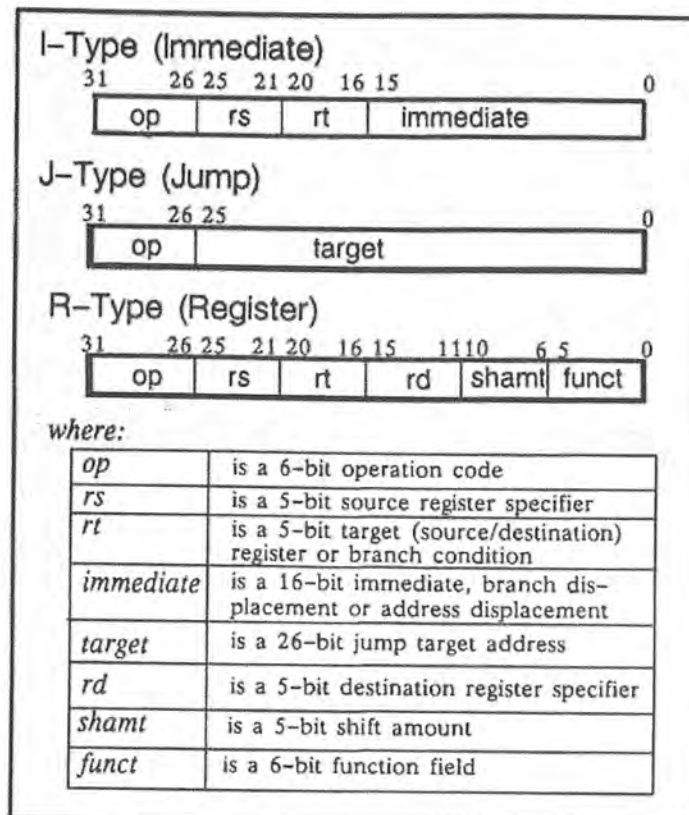


Figure A.1 R2000 Instruction Formats

Instruction Notation Conventions

In this appendix, all variable subfields in an instruction format (such as *rs*, *rt*, *immediate*, etc.) are shown in lower-case names.

For the sake of clarity, we sometimes use an alias for a variable subfield in the formats of specific instructions. For example, we use *rs* = *base* in the format for Load and Store instructions. Such an alias is always lower case, since it refers to a variable subfield.

In the instruction descriptions that follow, the *Operation* section describes the operation performed by each instruction using a high-level language notation. Special symbols used in the notation are described in Table A.2.

Symbol	Meaning
$\leftarrow$	Assignment
$\parallel$	Bit string concatenation
x^y	Replication of bit value x into a y -bit string. Note that x is always a single-bit value.
$x_{y..z}$	Selection of bits y through z of bit string x . Little-endian bit notation is always used. If y is less than z , this expression is an empty (zero length) bit string.
$+$	Two's complement addition
$-$	Two's complement subtraction
$*$	Two's complement multiplication
div	Two's complement integer division
mod	Two's complement modulo
$<$	Two's complement less than comparison
and	Bitwise logic AND
or	Bitwise logic OR
xor	Bitwise logic XOR
nor	Bitwise logic NOR
$GPR[x]$	General Register x . The contents of $GPR[0]$ is always zero. Attempts to alter the contents of $GPR[0]$ have no effect.
$CPR[z,x]$	Coprocessor unit z , general register x
$CCR[z,x]$	Coprocessor unit z , control register x
$T+i$	Indicates the time steps (CPU cycles) between operations. Thus, operations identified as occurring at $T+1$ are performed during the cycle following the one where the instruction was initiated. This type of operation occurs with loads, stores, jumps, branches and coprocessor instructions.
vAddress	Virtual address
pAddress	Physical address

Table A.2 R2000 Instruction Operation Notations

Instruction Notation Examples

The following examples illustrate the application of some of the instruction notation conventions:

Example #1:

$$\text{GPR}[rt] \leftarrow \text{immediate} \parallel 0^{16}$$

Sixteen zero bits are concatenated with an immediate value (typically 16 bits), and the 32-bit string (with the lower 16 bits set to zero) is assigned to General Purpose Register *rt*.

Example #2:

$$(\text{immediate}_{15})^{16} \parallel \text{immediate}_{15..0}$$

Bit 15 (the sign bit) of an immediate value is extended for 16 bit positions, and the result is concatenated with bits 15 through 0 of the immediate value to form a 32-bit sign-extended value.

Load and Store Instructions

All load operations have a latency of one instruction. That is, the instruction immediately following a load cannot use the contents of the register which will be loaded with the data being fetched from storage. An exception is that the target register for the load word left and load word right instructions may be specified as the same register used as the destination of a load instruction that immediately precedes it.

In the load/store operation descriptions, the functions listed in Table A.2a are used to summarize the handling of virtual addresses and physical memory.

Function	Description
Addr Translation	Uses the TLB to find the physical address given the virtual address. The function fails and an exception is taken if the entry for the page containing the virtual address is not present in the TLB (Translation Lookaside Buffer).
Load Memory	Uses the cache and main memory to find the contents of the word containing the specified physical address. The low-order two bits of the address and the access type field indicate which of each of the four bytes within the data word need to be returned. If the cache is enabled for this access, the entire word is returned and loaded into the cache.
Store Memory	Uses the cache, write buffer, and main memory to store the word or part of word specified as data into the word containing the specified physical address. The low-order two bits of the address and the access type field indicate which of the four bytes within the data word should be stored.

Table A.2a Load/Store Common Functions

The access type field indicates the size of the data item to be loaded or stored as shown in Table A.2b. Regardless of access type or byte-numbering order (endian-ness), the address specifies the byte which has the smallest byte address of the bytes in the addressed field. For a big-endian machine, this is the leftmost byte and contains the sign for a two's complement number; for a little-endian machine, this is the rightmost byte and contains the lowest precision byte.

Access Type		Meaning
Mnemonic	Value	
WORD	3	word (32 bits)
TRIPLE-BYTE	2	triple-byte (24 bits)
HALFWORD	1	halfword (16 bits)
BYTE	0	byte (8 bits)

Table A.2b Access Type Specifications for Loads/Stores

The bytes within the addressed word which are used can be determined directly from the access type and the two low-order bits of the address, as shown in Table A.2c. Note that certain combinations of access type and low-order address bits can never occur (WORD/01/10/11, TRIPLE-BYTE/10/11, HALFWORD/01/11).

Access Type 1 0	Low-Order Address Bits 1 0	Bytes Accessed	
		31 _____ 0 Big-Endian	31 _____ 0 Little-Endian
1 1 (word)	0 0	0 1 2 3	3 2 1 0
1 0 (triple-byte)	0 0	0 1 2	2 1 0
	0 1	1 2 3	3 2 1
0 1 (halfword)	0 0	0 1	1 0
	1 0	2 3	3 2
0 0 (byte)	0 0	0	0
	0 1	1	1
	1 0	2	2
	1 1	3	3

Table A.2c Byte Specifications for Loads/Stores

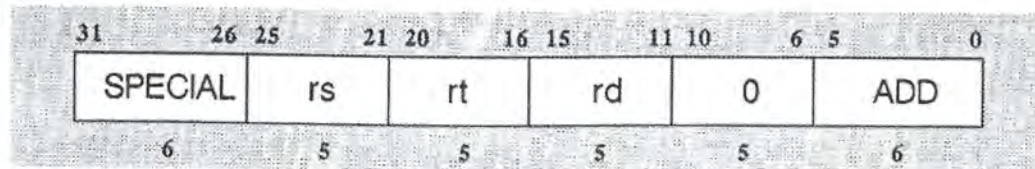
Jump and Branch Instructions

All jump and branch instructions are implemented with a delay of exactly one instruction. That is, the instruction immediately following a jump or branch (i.e., occupying the delay slot) is always executed while the target instruction is being fetched from storage. It is not valid for a delay slot to be occupied itself by a jump or branch instruction; however, this error is not detected, and the results of such an operation are undefined.

If an exception or interrupt prevents the completion of a legal instruction during a delay slot, the R2000 sets the EPC register to point at the jump or branch instruction which precedes it. When the code is restarted, both the jump or branch instructions and the instruction in the delay slot are re-executed.

Because jump and branch instructions may be restarted after exceptions or interrupts, they must be restartable. Therefore, when a jump or branch instruction stores a return link value, register 31 may not be used as a source register.

Since instructions must be word-aligned, a *Jump Register* or *Jump and Link Register* instruction must use a register whose two low-order bits are zero. If these low order bits are not zero, an address exception will occur when the jump target instruction is subsequently fetched.

ADD**ADD****Format:**

ADD rd,rs,rt

Description:

The contents of general register *rs* and the contents of general register *rt* are added to form a 32-bit result. The result is placed into general register *rd*.

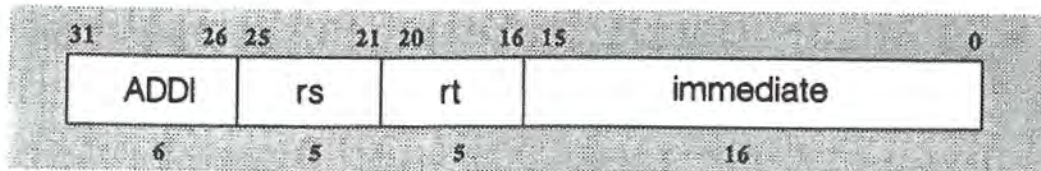
An overflow exception occurs if the two highest order carry-out bits differ (two's complement overflow).

Operation:

T:	$\text{GPR}[\text{rd}] \leftarrow \text{GPR}[\text{rs}] + \text{GPR}[\text{rt}]$
----	--

Exceptions:

Overflow exception

ADDI**Add Immediate****Format:**

ADDI rt,rs,immediate

Description:

The 16-bit *immediate* is sign-extended and added to the contents of general register *rs* to form a 32-bit result. The result is placed into general register *rt*.

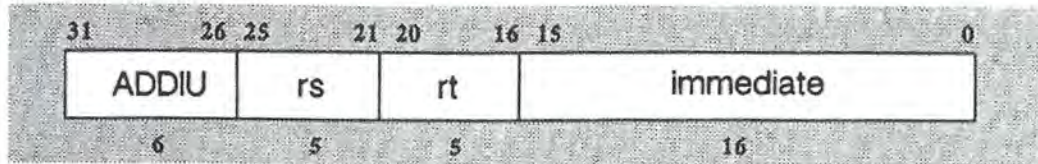
An overflow exception occurs if the two highest order carry-out bits differ (two's complement overflow).

Operation:

$$T: \quad \text{GPR}[rt] \leftarrow \text{GPR}[rs] + (\text{immediate}_{15})^{16} \parallel \text{immediate}_{15..0}$$

Exceptions:

Overflow exception

Add Immediate Unsigned**ADDIU****Format:**

ADDIU rt,rs,immediate

Description:

The 16-bit *immediate* is sign-extended and added to the contents of general register *rs* to form a 32-bit result. The result is placed into general register *rt*. No overflow exception occurs under any circumstances.

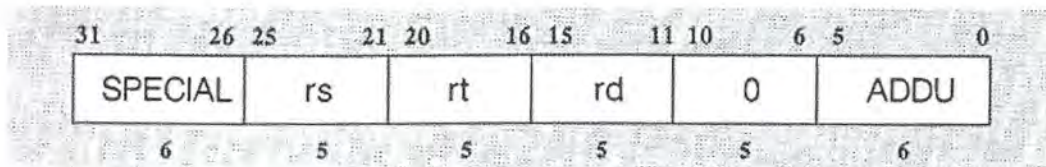
Note that the only difference between this instruction and the ADDI instruction is that ADDIU never causes an overflow exception.

Operation:

$$T: \quad \text{GPR}[rt] \leftarrow \text{GPR}[rs] + (\text{immediate}_{15})^{16} \parallel \text{immediate}_{15..0}$$

Exceptions:

None.

ADDU**ADD Unsigned****Format:**

ADDU rd,rs,rt

Description:

The contents of general register *rs* and the contents of general register *rt* are added to form a 32-bit result. The result is placed into general register *rd*.

No overflow exception occurs under any circumstances.

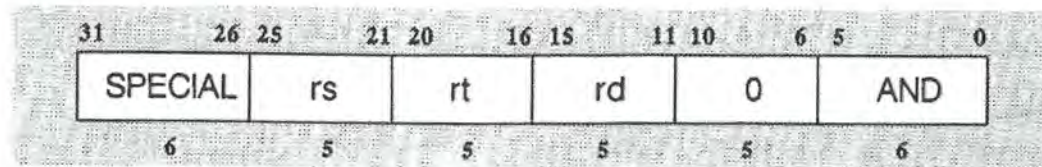
Note that the only difference between this instruction and the ADD instruction is that ADDU never causes an overflow exception.

Operation:

T:	$\text{GPR}[\text{rd}] \leftarrow \text{GPR}[\text{rs}] + \text{GPR}[\text{rt}]$
----	--

Exceptions:

None.

And**AND****Format:**

AND rd,rs,rt

Description:

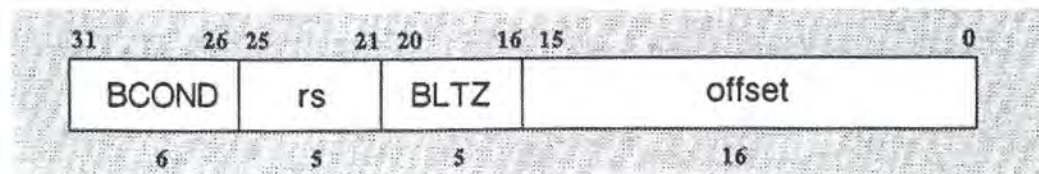
The contents of general register *rs* are combined with the contents of general register *rt* in a bit-wise logical AND operation. The result is placed into general register *rd*.

Operation:

T: $\text{GPR}[\text{rd}] \leftarrow \text{GPR}[\text{rs}] \text{ and } \text{GPR}[\text{rt}]$

Exceptions:

None.

BLTZ**Branch On Less Than Zero****Format:**

BLTZ rs,offset

Description:

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset*, shifted left two bits and sign-extended to 32 bits. If the contents of general register *rs* have the sign bit set, then the program branches to the target address, with a delay of one instruction.

Operation:

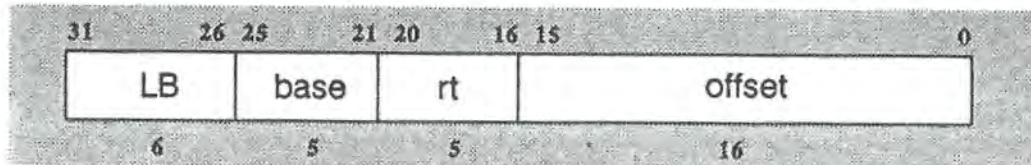
```

T:  target ← (offset15)14 || offset || 02
    condition ← (GPR[rs]31 = 1)
T + 1: if condition then
        PC ← PC + target
    endif

```

Exceptions:

None.

Load Byte**LB****Format:**

LB rt,offset(base)

Description:

The 16-bit *offset* is sign-extended and added to the contents of general register *base* to form a 32-bit unsigned effective address. The contents of the byte at the memory location specified by the effective address are sign-extended and loaded into general register *rt*.

Operation:

```

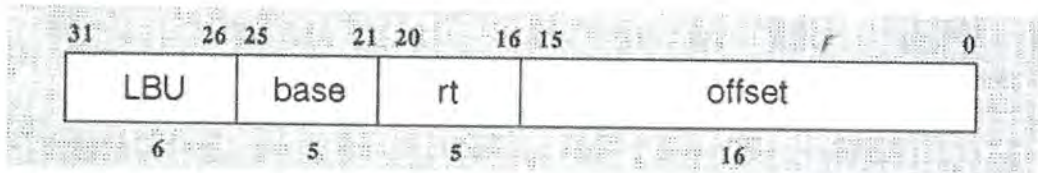
T:   vAddress ← (offset15)16 || offset15..0 + GPR[base]
      (pAddress, nonCacheable) ← AddrTranslation (vAddress)
      mem ← LoadMemory (nonCacheable, BYTE, pAddress)
      byte ← vAddress1..0

T + 1: if BigEndian then
      GPR[rt] ← (mem31-8*byte)24 || mem31-8*byte .. 24-8*byte
    else
      GPR[rt] ← (mem7+8*byte)24 || mem7+8*byte .. 8*byte
    endif

```

Exceptions:

UTLB miss fault
 TLB miss fault
 Bus error exception
 Address error exception

LBU**Load Byte Unsigned****Format:**LBU *rt*,*offset*(*base*)**Description:**

The 16-bit *offset* is sign-extended and added to the contents of general register *base* to form a 32-bit unsigned effective address. The contents of the byte at the memory location specified by the effective address are zero-extended and loaded into general register *rt*.

Operation:

```

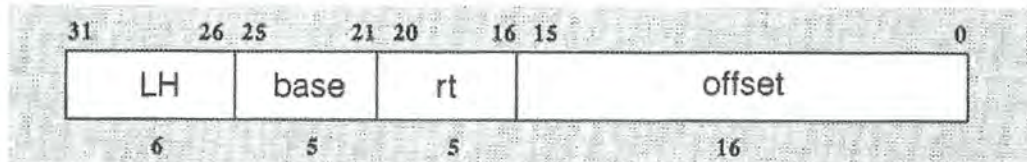
T:   vAddress ← (offset15)16 || offset15:0 + GPR[base]
      (pAddress, nonCacheable) ← AddrTranslation (vAddress)
      mem ← LoadMemory (nonCacheable, BYTE, pAddress)
      byte ← vAddress1:0

T + 1: if BigEndian then
      GPR[rt] ← 024 || mem31-8*byte..24-8*byte
    else
      GPR[rt] ← 024 || mem7+8*byte..8*byte
    endif

```

Exceptions:

UTLB miss fault
 TLB miss fault
 Bus error exception
 Address error exception

Load Halfword**LH****Format:**

LH rt,offset(base)

Description:

The 16-bit *offset* is sign-extended and added to the contents of general register *base* to form a 32-bit unsigned effective address. The contents of the halfword at the memory location specified by the effective address are sign-extended and loaded into general register *rt*.

If the least significant bit of the effective address is non-zero, an address error exception occurs.

Operation:

```

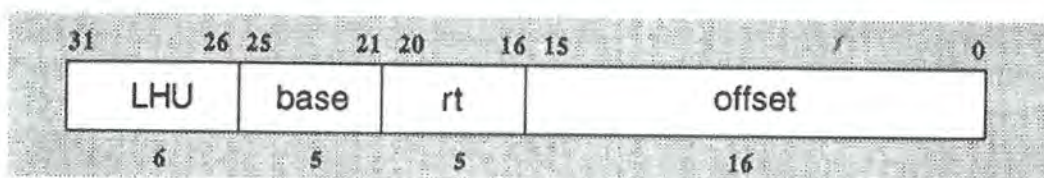
T:    vAddress ← (offset15)16 || offset15:0 + GPR[base]
      (pAddress, nonCacheable) ← AddrTranslation (vAddress)
      mem ← LoadMemory (nonCacheable, HALFWORD, pAddress)
      byte ← vAddress1:0

T + 1: if BigEndian then
      GPR[rt] ← (mem31-8*byte)16 || mem31-8*byte..16-8*byte
    else
      GPR[rt] ← (mem15+8*byte)16 || mem15+8*byte..8*byte
    endif

```

Exceptions:

UTLB miss fault
 TLB miss fault
 Bus error exception
 Address error exception

LHU**Load Halfword Unsigned****Format:**LHU *rt*,*offset*(*base*)**Description:**

The 16-bit *offset* is sign-extended and added to the contents of general register *base* to form a 32-bit unsigned effective address. The contents of the halfword at the memory location specified by the effective address are zero-extended and loaded into general register *rt*.

If the least significant bit of the effective address is non-zero, an address error exception occurs.

Operation:

```

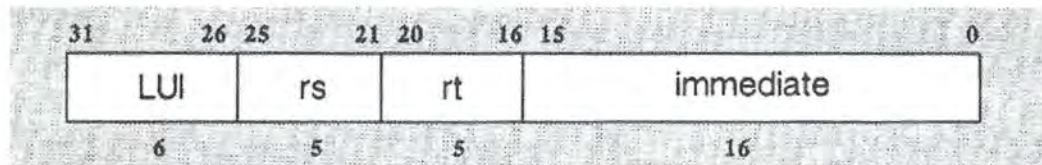
T:   vAddress ← (offset15)16 || offset15..0 + GPR[base]
      (pAddress, nonCacheable) ← AddrTranslation (vAddress)
      mem ← LoadMemory (nonCacheable, HALFWORD, pAddress)
      byte ← vAddress1..0

T + 1: if BigEndian then
      GPR[rt] ← 016 || mem31-8*byte..16-8*byte
    else
      GPR[rt] ← 016 || mem15+8*byte..8*byte
    endif

```

Exceptions:

UTLB miss fault
 TLB miss fault
 Bus error exception
 Address error exception

Load Upper Immediate**LUI****Format:**LUI *rt*,*immediate***Description:**

The 16-bit *immediate* is shifted left 16 bits and concatenated to 16 bits of zeroes. The result is placed into general register *rt*.

Operation:

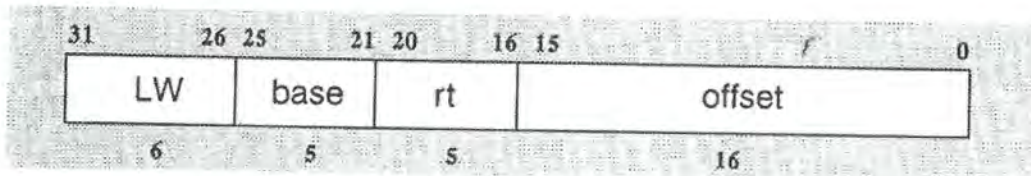
T:	$\text{GPR}[\text{rt}] \leftarrow \text{immediate} \parallel 0^{16}$
----	--

Exceptions:

None.

LW

Load Word

**Format:**LW *rt*,*offset*(*base*)**Description:**

The 16-bit *offset* is sign-extended and added to the contents of general register *base* to form a 32-bit unsigned effective address. The contents of the word at the memory location specified by the effective address are loaded into general register *rt*.

If either of the two least significant bits of the effective address is non-zero, an address error exception occurs.

Operation:

```

T:   vAddress ← (offset15)16 || offset15:0 + GPR[base]
      (pAddress, nonCacheable) ← AddrTranslation (vAddress)
      mem ← LoadMemory (nonCacheable, WORD, pAddress)
      byte ← vAddress1:0
T+1: GPR[rt] ← mem;

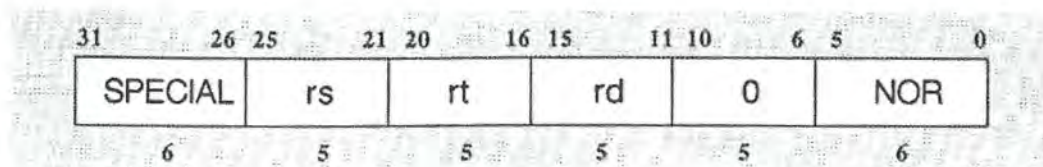
```

Exceptions:

UTLB miss fault
 TLB miss fault
 Bus error exception
 Address error exception

NOR

Nor

**Format:**

NOR rd,rs,rt

Description:

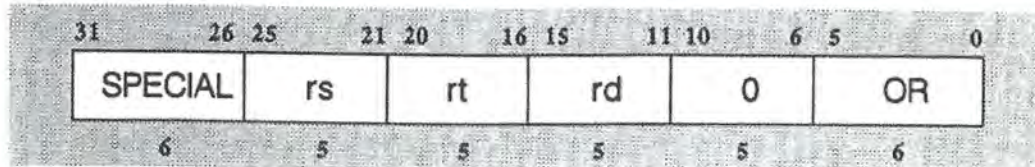
The contents of general register *rs* are combined with the contents of general register *rt* in a bit-wise logical NOR operation. The result is placed into general register *rd*.

Operation:

T: $\text{GPR}[rd] \leftarrow \text{GPR}[rs] \text{ nor } \text{GPR}[rt]$

Exceptions:

None.

Or**OR****Format:**

OR rd,rs,rt

Description:

The contents of general register *rs* are combined with the contents of general register *rt* in a bit-wise logical OR operation. The result is placed into general register *rd*.

Operation:

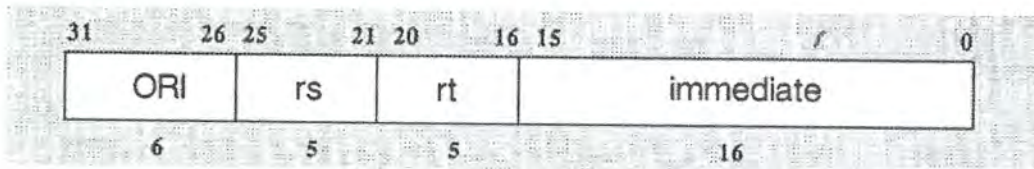
T:	$\text{GPR}[\text{rd}] \leftarrow \text{GPR}[\text{rs}] \text{ or } \text{GPR}[\text{rt}]$
----	--

Exceptions:

None.

ORI

Or Immediate

**Format:**ORI *rt*,*rs*,*immediate***Description:**

The 16-bit *immediate* is zero-extended and combined with the contents of general register *rs* in a bit-wise logical OR operation. The result is placed into general register *rt*.

Operation:

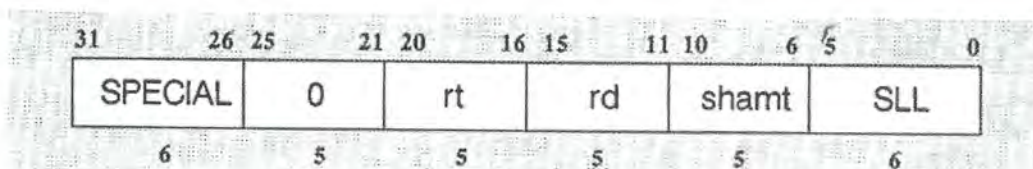
$$T: \quad \text{GPR}[rt] \leftarrow \text{GPR}[rs]_{31..16} \parallel (\text{immediate or } \text{GPR}[rs]_{15..0})$$

Exceptions:

None.

SLL

Shift Left Logical

**Format:**

SLL rd,rt,shamt

Description:

The contents of general register *rt* are shifted left by *shamt* bits, inserting zeroes into the low order bits. The 32-bit result is placed in register *rd*.

Operation:

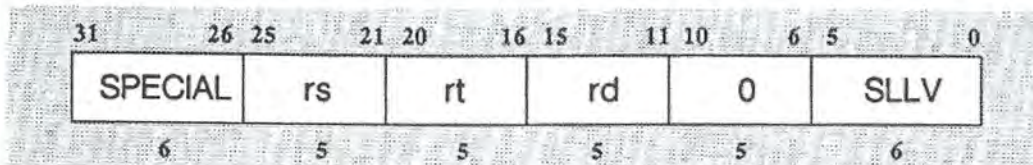
$$T: \quad \text{GPR}[rd] \leftarrow \text{GPR}[rt]_{31-shamt,0} \parallel 0_{shamt}$$

Exceptions:

None.

Shift Left Logical Variable

SLLV



Format:

SLLV rd,rt,rs

Description:

The contents of general register *rt* are shifted left by the number of bits specified by the low-order 5 bits of the contents of general register *rs*, inserting zeroes into the low order bits. The 32-bit result is placed in register *rd*.

Operation:

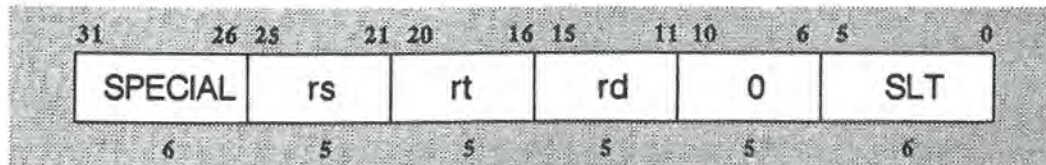
$$T: \quad \text{GPR}[rd] \leftarrow \text{GPR}[rt]_{(31-\text{GPR}[rs]_{4..0})..0} \parallel 0_{\text{GPR}[rs]_{4..0}}$$

Exceptions:

None.

SLT

Set On Less Than

**Format:**

SLT rd,rs,rt

Description:

The contents of general register *rt* are compared with the contents of general register *rs*. Considering both quantities as signed 32-bit integers, if the contents of general register *rs* are less than the contents of general register *rt*, the result is set to one, otherwise the result is set to zero. The result is placed into general register *rd*.

No overflow exception occurs under any circumstances. The comparison is valid even if the subtraction used during the comparison overflows.

Operation:

```

T:      if GPR[rs] < GPR[rt] then
          GPR[rd] ← 031 || 1
        else
          GPR[rd] ← 032
        endif

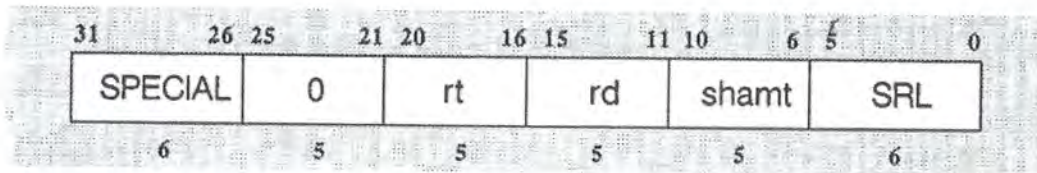
```

Exceptions:

None.

SRL

Shift Right Logical

**Format:**

SRL rd,rt,shamt

Description:

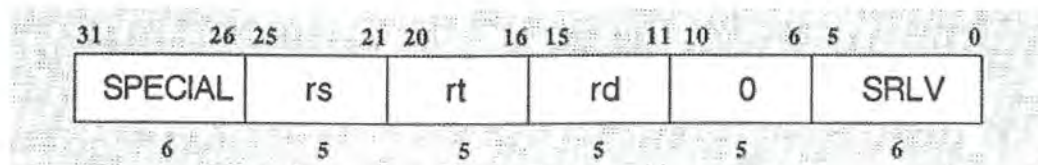
The contents of general register *rt* are shifted right by *shamt* bits, inserting zeroes into the high order bits. The 32-bit result is placed in register *rd*.

Operation:

$$T: \quad \text{GPR}[rd] \leftarrow 0^{\text{shamt}} \parallel \text{GPR}[rt]_{31..shamt}$$

Exceptions:

None.

Shift Right Logical Variable**SRLV****Format:**

SRLV rd,rt,rs

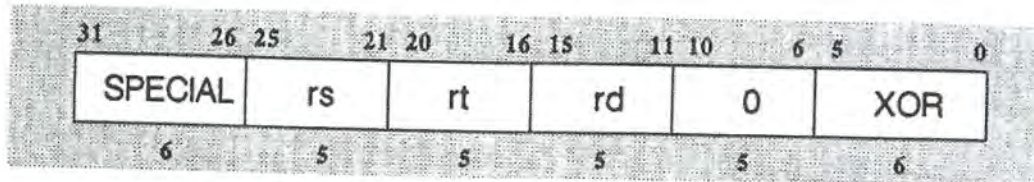
Description:

The contents of general register *rt* are shifted right by the number of bits specified by the low-order 5 bits of the contents of general register *rs*, inserting zeroes into the high order bits. The 32-bit result is placed in register *rd*.

Operation:

$$T: \quad \text{GPR}[rd] \leftarrow 0^{\text{GPR}[rs]_{4..0}} \parallel \text{GPR}[rt]_{31..(\text{GPR}[rs]_{4..0})}$$
Exceptions:

None.

Exclusive Or**XOR****Format:**

XOR rd,rs,rt

Description:

The contents of general register *rs* are combined with the contents of general register *rt* in a bit-wise logical exclusive OR operation. The result is placed into general register *rd*.

Operation:

T: $\text{GPR}[rd] \leftarrow \text{GPR}[rs] \text{ xor } \text{GPR}[rt]$

Exceptions:

None.

C

Machine Language Programming Tips

The RISC architecture implemented by the R2000 Processor provides an efficient, uniform, and streamlined instruction set to obtain maximum efficiency for the most commonly performed operations. As a result, some operations that would require single, multi-cycle instructions in more traditional architectures require multiple, single-cycle R2000 instructions.

The R2000 architecture also provides no condition code register containing status bits such as *Carry* and *Overflow*. Instead, the conditions generated by the Set instructions (*slt*, *sltu*) are loaded directly into a general-purpose register. This approach greatly simplifies handling of the instruction pipeline and eases many compiler tasks but does require that programs explicitly check for conditions such as overflow and carry.

This appendix describes techniques that can be used to implement the following operations:

- Handling 32-bit addresses or constants
- Implementing indexed addressing
- Using the Jump Register (*jr*) instruction for subroutine returns
- Jumping to 32-bit addresses
- Branching on arithmetic comparisons
- Filling the branch delay slot
- Testing for overflow
- Testing for carry
- Performing multi-precision math

Note that the MIPS assembly language supports many of these functions (such as loading 32-bit addresses and branching on arithmetic comparisons) that are not

directly implemented by the R2000 machine language. It does this using techniques similar to those described in this appendix. Refer to **Appendix D** for an overview of the MIPS assembly language.

In many of the following examples, a temporary register is used to hold intermediate results. In the following description, `$at` is used to represent that temporary register, and it is assumed that the register is reserved for just this purpose and therefore doesn't conflict with anything else.

32-bit Addresses or Constants

The R2000 does not provide specific "load address" or "load immediate" instructions. (Note: the MIPS assembly language does provide load address (la) and load immediate (li) instructions that are implemented using multiple machine language instructions. To load an address with relocatable code, you *must* use the assembler's la instruction.) Use the following two-instruction sequence to load any 32-bit pattern into a register:

```
lui $destination, <upper 16 bits>
ori $destination, <lower 16 bits>
```

There are three special cases which require only one instruction to obtain the desired 32-bit pattern:

- **Constants with the upper 16 bits set to zero.** Use the `ori` instruction, which zero-extends the immediate field:

```
ori $destination, $0, <lower 16 bits>
```

- **Constants with the upper 17 bits set to one.** Use the low-order 16 bits of the constant in the `addi` instruction, which sign-extends the immediate field:

```
addi $destination, $0, <lower 16 bits>
```

- **Constants with the lower 16 bits set to zero.** Use the high-order 16 bits of the constant in the `lui` instruction, which shifts its immediate field left by 16 bits, bringing in zeroes on the right:

```
lui $destination, <upper 16 bits>
```

Testing for Overflow

The R2000 does not provide a status bit to indicate whether an arithmetic operation resulted in an overflow. The signed addition and subtraction instructions (**ADD** and **SUB**) trap if an overflow occurs and thus implement overflow detection at no cost. However, if it is necessary to detect signed overflow without using traps or to detect overflow for unsigned operations, the techniques described in this section can be used.

Figure C.1 provides examples of code that checks for overflow for various arithmetic operations. These examples are based on the following simple rules for signed overflow:

- During addition, overflow occurs if the signs of the addends are the same and the sign of the sum is different.
- During subtraction, overflow occurs if the signs of the operands are different and the sign of the resultant difference is not the same as the sign of the minuend.

Signed Addition

```

/* compute t0 = t1 + t2, branch to L on signed overflow */
addu  t0, t1, t2  /* compute sum */
xor   t3, t1, t2  /* if operands have different signs */
bltz  t3, 1f      /* then overflow not possible */
/* t1 and t2 have same sign */
xor   t3, t0, t1  /* if sum does not also have the same sign */
bltz  t3, L       /* then addition overflowed */
/* nop */
1:

```

Signed Subtraction

```

/* compute t0 = t1 - t2, branch to L on signed overflow */
subu  t0, t1, t2  /* compute difference */
xor   t3, t1, t2  /* if operands have same signs */
bgez  t3, 1f      /* then overflow not possible */
/* t1 and t2 have different signs */
xor   t3, t0, t1  /* if difference does not also have */
/* the same sign as the minuend */
bltz  t3, L       /* then subtraction overflowed */
/* nop */
1:

```

Unsigned Addition

```

/* compute t0 = t1 + t2 , branch to L on unsigned overflow */
addu  t0, t1, t2
not   t3, t1
sltu  t3, t3, t2
bne   t3, 0, L

```

Unsigned Subtraction

```

/* compute t0 = t1 - t2, branch to L on unsigned overflow */
subu  t0, t1, t2
sltu  t3, t1, t2
bne   t3, 0, L

```

Signed Multiplication

```

/* compute t0 = t1 * t2, branch to L on signed overflow */
mult  t1, t2
mflo  t0
mfhi  t3
sra   t4, t0, 31
bne   t4, t3, L

```

Unsigned Multiplication

```

/* compute t0 = t1 * t2, branch to L on unsigned overflow */
multu t1, t2
mflo  t0
mfhi  t3
bne   t3, 0, L      /* if HI  $\neq$  0 ,then multiplication overflowed */

```

Figure C.1 Calculating Overflow for Arithmetic Operations

D

Assembly Language Programming

This appendix provides an overview of the assembly language supported by the MIPS compiler system. The assembler converts assembly language statements into machine code. In most assembly languages, each instruction corresponds to a single machine instruction; however, some MIPS assembly language instructions can generate several R2000 machine instructions. This approach provides a more regular assembler that generates optimized code for certain short sequences and also results in assembly programs that can run without modification on future machines, which might have extended machine instructions. See *Basic Machine Definition* at the end of this appendix for more information about assembler instructions that generate multiple machine instructions.

Register Use and Linkage

This section describes the naming and usage conventions that the assembler applies to the R2000 and R2010 registers.

General Registers

The R2000 Processor has thirty-two 32-bit integer registers. Table D.1 summarizes the assembler's usage conventions and restrictions for these registers. The assembler reserves all register names, and you must use lowercase for the names. All register names start with a dollar sign (\$).

The general registers have the names \$0..\$31. By including the file *regdef.h* (use `#include <regdef.h>`) in your program, you can use software names for some general registers. The operating system and the assembler use the general registers \$1, \$26, \$27, \$28, and \$29 for specific purposes. (NOTE: Attempts to use these general registers in other ways can produce unexpected results.) If a program uses the names \$1, \$26, \$27, \$28, \$29 rather than the names \$at, \$kt0, \$kt1, \$gp, \$sp respectively, the assembler issues warning messages.

register name	software name (from regdef.h)	use and linkage
\$0		always has the value 0
\$at or \$1		reserved for the assembler
\$2..\$3	v0-v1	used for expression evaluations and to hold integer function results. Also used to pass the static link when calling nested procedures.
\$4..\$7	a0-a3	used to pass the first 4 words of integer type actual arguments; their values are not preserved across procedure calls
\$8..\$15	t0-t7	temporary registers, used for expression evaluations; their values are not preserved across procedure calls.
\$16..\$23	s0-s7	saved registers; their values must be preserved across procedure calls.
\$24..\$25	t8-t9	temporary registers, used for expression evaluations; their values are not preserved across procedure calls.
\$26..\$27 or \$kt0..\$kt1	k0-k1	reserved for the operating system kernel
\$28 or \$gp	gp	contains the global pointer
\$29 or \$sp	sp	contains the stack pointer
\$30	s8	a saved register (like s0-s7)
\$31	ra	contains the return address; used for expression evaluation.

Table D.1 General (Integer) Registers

General register \$0 always contains the value 0. All other general registers are equivalent, except that general register \$31 also serves as the implicit link register for jump and link instructions.

mips R2000 RISC ARCHITECTURE

GERRY KANE

This book is the primary architectural reference manual for the R2000 family of processors. It describes the functional characteristics and capabilities of the R2000 Processor and the R2010 Floating Point Accelerator (FPA). These processors are available from the following manufacturers:



Integrated Device Technology, Inc.

- Integrated Device Technology, Inc.
3226 Scott Boulevard
P.O. Box 58015
Santa Clara, CA 95054-3029
(408) 727-6116
Telex 887766
FAX (408) 988-3029

LSI LOGIC



- LSI Logic Corporation
1551 McCarthy Blvd.
MS D-105
Milpitas, CA 95035
(408) 433-8000
Telex 171.641
FAX (408) 433-3816
Attn: Standard Products Marketing

mips

- MIPS Computer Systems, Inc.
930 Arques Avenue
Sunnyvale, CA 94086-3650
(408) 720-1700
Telex 510 601 5346
FAX (408) 720-9809



- Performance Semiconductor Corporation
610 E. Weddell Drive
Sunnyvale, CA 94089
(408) 734-9000
FAX (408) 734-0258
Attn: Microprocessor Marketing

Prentice Hall
Englewood Cliffs, N.J. 07632

ISBN 0-13-584749-4





5 icons: bookmark, quote, share, etc.

MIPS RISC architecture : R2000/R3000

★★★★★ [0 reviews](#)

Author: [Gerry Kane](#)

Print Book, English, 1987

Edition: [View all formats and editions](#)

Publisher: Prentice-Hall, Englewood Cliffs, NJ, 1987

Physical Description:	1 Band (Mehrere Zählungen) : Illustrationen : 24 cm
ISBN:	9780135847497, 9780135842935, 0135847494,... Show more ▾
OCLC Number / Unique Identifier:	728453247
Subjects:	C.1 PROCESSOR ARCHITECTURES

[Show less information](#) ▴

Find a Copy at a Library

Filter by: [Any format](#) ▾ [Any edition](#) ▾ [Distance within 200+ mi](#) ▾

9 editions in 272 libraries

[Featured libraries](#) [All libraries](#)

Showing 190 libraries near [Washington, District of Columbia](#) ☐ Only public libraries

Regional or National



[Library of Congress](#)

2.2 miles from your current location.

101 Independence Avenue, SE
Washington, DC 20540
United States

[Get Directions](#)



Favorite Library

[Borrow](#)



MIPS RISC architecture

Author: [Gerry Kane](#)

Print Book, English, ©1989

Edition: [View all formats and editions](#)

Publisher: Prentice Hall, Englewood Cliffs, NJ, ©1989

Physical Description: 1 volume (various
pagings) :
illustrations ; 24
cm

ISBN: 9780135842935,
9780135847497,
013584293X,
0135847494
[Show less](#) ^

OCLC Number / Unique Identifier: 1087058788

Subjects: [Computer
architecture
MIPS R2000
series
microprocessor...](#)
[Show more](#) v

Notes: Includes index

[Show less information](#) ^

**Find a Copy at a
Library**

Filter by: Any format v Any edition v Distance within 200+ mi v

Screen capture completed by J L Mullins from WorldCat on November 21, 2022

 **9 editions in 272 libraries**

Featured libraries All libraries

Showing 190 libraries near [Washington, District of Columbia](#) ☐ Only public libraries

Regional or National



[Library of Congress](#)

 **2.2 miles** from your current location.

101 Independence Avenue, SE
Washington, DC 20540
United States

[Get Directions](#) 


Favorite Library

Borrow 

LIBRARY

CATALOG

BOOK

MIPS R2000 RISC architecture

Full Record

MARC Tags

Personal name

Kane, Gerry, 1942-

Main title

MIPS R2000 RISC architecture / Gerry Kane.

Published/Created

Englewood Cliffs, NJ : Prentice Hall, c1987.

Request this Item

 LC Find It

More Information

>

LCCN Permalink	https://lcn.loc.gov/88060290
Description	1 v. (various pagings) : ill. ; 24 cm.
ISBN	0135847494 (pbk.)
LC classification	QA76.8.M52 K36 1987
LC Subjects	MIPS R2000 series microprocessors. Computer architecture. Reduced instruction set computers.

Browse by shelf order	QA76.8.M52
Notes	Includes index.
LCCN	88060290
Dewey class no.	004.165
Type of material	Book

Item Availability

>

CALL NUMBER	QA76.8.M52 K36 1987 LANDOVR Copy 1
Request in	Jefferson or Adams Building Reading Rooms - STORED OFFSITE
Status	Not Charged

CALL NUMBER	QA76.8.M52 K36 1987 FT MEADE Copy 2
Request in	Jefferson or Adams Building Reading Rooms - STORED OFFSITE
Status	Not Charged

LIBRARY | CATALOG

← 1 of 1 →

BOOK

MIPS R2000 RISC architecture

[Full Record](#)

[MARC Tags](#)

000	00914cam a2200277 a 4500
001	3858067
005	19890914113002.8
008	880201s1987 njua 001 0 eng
035	— 9 (DLC) 88060290
906	— a 7 b cbc c orignew d 2 e opcn f 19 g y-gencatlg
955	— a by 13 to bt00 06-15-88; fg09 06-28-88
010	— a 88060290
020	— a 0135847494 (pbk.)
040	— a DLC c DLC d DLC
050	00 a QA76.8.M52 b K36 1987
082	00 a 004.165 2 19
100	1_ a Kane, Gerry, d 1942-
245	10 a MIPS R2000 RISC architecture / c Gerry Kane.
260	— a Englewood Cliffs, NJ : b Prentice Hall, c c1987.
300	— a 1 v. (various pagings) : b ill. ; c 24 cm.
504	— a Includes index.
650	_0 a MIPS R2000 series microprocessors.
650	_0 a Computer architecture.
650	_0 a Reduced instruction set computers.
922	— a co
991	— b c-GenColl h QA76.8.M52 i K36 1987 p 00003841078 t Copy 1 w BOOKS

Request this Item

 LC Find It

Item Availability

>

CALL NUMBER	QA76.8.M52 K36 1987 LANDOVR Copy 1
Request in	Jefferson or Adams Building Reading Rooms - STORED OFFSITE
Status	Not Charged

CALL NUMBER	QA76.8.M52 K36 1987 FT MEADE Copy 2
Request in	Jefferson or Adams Building Reading Rooms - STORED OFFSITE
Status	Not Charged



QA 76
.9
.A73
C74
2000
COPY 2

FT MEADE
GenCo 11

Cragon

Computer Architecture and Implementation

CAMBRIDGE

This textbook provides a clear and concise introduction to computer architecture and implementation. Two important themes are interwoven throughout the book. The first is an overview of the major concepts and design philosophies of computer architecture and organization. The second is the early introduction and use of analytic modeling of computer performance.

The author begins by describing the classic von Neumann architecture and then presents in detail a number of performance models and evaluation techniques. He goes on to cover user instruction set design, including RISC architecture. A unique feature of the book is its memory-centric approach – memory systems are discussed before processor implementations. The author also deals with pipelined processors, input/output techniques, queuing modes, and extended instruction set architectures. Each topic is illustrated with reference to actual IBM and Intel architectures.

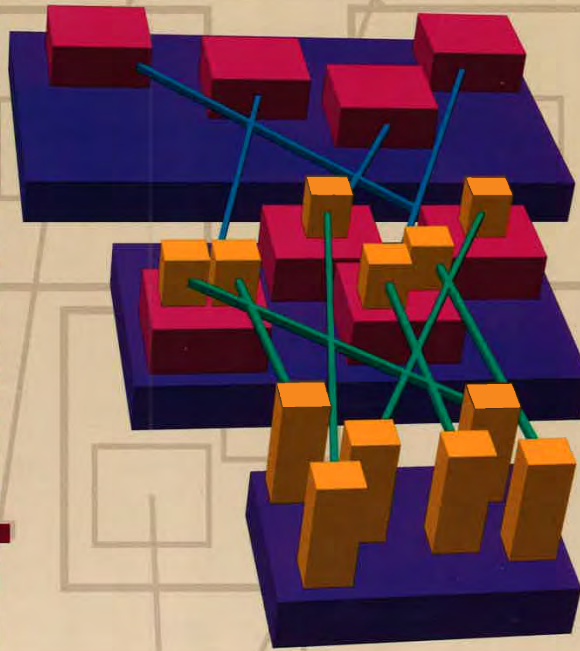
The book contains many worked examples and over 130 homework exercises. It is an ideal textbook for a one-semester undergraduate course in computer architecture and implementation.

Harvey G. Cragon holds the Ernest J. Cockrell, Jr., Centennial Chair Emeritus in Engineering at the University of Texas at Austin. He is a member of the U.S. National Academy of Engineering, a Life Fellow of the IEEE, and a Fellow of the Association for Computing Machinery. He has over 30 years' industrial experience, including 25 years with Texas Instruments. He has written over 50 technical papers and holds 9 patents in the field of computer engineering.

CAMBRIDGE
UNIVERSITY PRESS



Computer Architecture and Implementation



Harvey G. Cragon

COMPUTER ARCHITECTURE AND IMPLEMENTATION

HARVEY G. CRAGON

University of Texas at Austin



QA76
.9
.A73C74
2000
Copy 2

PUBLISHED BY THE PRESS SYNDICATE OF THE UNIVERSITY OF CAMBRIDGE
The Pitt Building, Trumpington Street, Cambridge, United Kingdom

CAMBRIDGE UNIVERSITY PRESS

The Edinburgh Building, Cambridge CB2 2RU, UK <http://www.cup.cam.ac.uk>
40 West 20th Street, New York, NY 10011-4211, USA <http://www.cup.org>
10 Stamford Road, Oakleigh, Melbourne 3166, Australia
Ruiz de Alarcón 13, 28014 Madrid, Spain

© Cambridge University Press 2000

This book is in copyright. Subject to statutory exception
and to the provisions of relevant collective licensing agreements,
no reproduction of any part may take place without
the written permission of Cambridge University Press.

First published 2000

Printed in the United States of America

Typeface Stone Serif 9.25/13 pt. System $\text{\TeX}$ 2_ε [TB]

A catalog record for this book is available from the British Library.

Library of Congress Cataloging-in-Publication Data

Cragon, Harvey G.

Computer architecture and implementation / Harvey G. Cragon.

p. cm.

ISBN 0-521-65168-9 (hard.)

1. Computer architecture. I. Title.

QA76.9.A73C74 2000

004.2'2—dc21

99-16243

CIP

ISBN 0 521 65168 9 hardback



CONTENTS

<i>Preface</i>	<i>page viii</i>
1 COMPUTER OVERVIEW	1
1.0 Introduction	1
1.1 von Neumann Model	2
1.2 The von Neumann Architecture	5
1.2.1 The von Neumann Instruction Set Architecture	6
1.2.2 Instruction Interpretation Cycle	10
1.2.3 Limitations of the von Neumann Instruction Set Architecture	13
1.3 Historical Notes	13
1.3.1 Precursors to the von Neumann Instruction Set Architecture	14
REFERENCES	23
EXERCISES	24
2 PERFORMANCE MODELS AND EVALUATION	26
2.0 Introduction	26
2.1 Performance Models	26
2.2 Amdahl's Law	37
2.3 Moore's Law	39
2.4 Learning Curve	42
2.5 Grosch's Law	44
2.6 Steady-State Performance	45
2.7 Transient Performance	46
REFERENCES	47
EXERCISES	47
3 USER INSTRUCTION SET DESIGN	50
3.0 Introduction	50
3.1 Lexical Level	51

3.2	Instruction Set Architecture	51
3.2.1	Addresses	53
3.2.2	Operations	62
3.2.3	Data Types	74
3.2.3.1	User Data Types	75
3.2.3.2	Program Sequencing Data Types	88
3.2.4	Instruction Composition	93
3.3	CISC and RISC Architectural Styles	98
3.4	Static and Dynamic Instruction Statistics	103
3.5	Arithmetic	109
3.5.1	Addition/Subtraction	111
3.5.2	Multiplication and Division	114
3.5.3	Floating-Point Arithmetic	118
3.5.3.1	Precision Treatment	121
	REFERENCES	127
	EXERCISES	128
4	MEMORY SYSTEMS	131
4.0	Introduction	131
4.1	Hierarchical Memory	132
4.2	Paged Virtual Memory	133
4.3	Caches	145
4.4	Interleaved Real Memory	163
4.5	Virtual- and Real-Address Caches	171
4.6	Segmented Virtual Memory	173
4.7	Disk Memory	176
4.8	System Operation	179
	REFERENCES	181
	EXERCISES	181
5	PROCESSOR CONTROL DESIGN	183
5.0	Introduction	183
5.1	Hardwired Control	185
5.2	Microprogrammed Control	193
	REFERENCES	201
	EXERCISES	202
6	PIPELINED PROCESSORS	204
6.0	Introduction	204
6.1	Performance Models	205
6.2	Pipeline Partitioning	207
6.3	Pipeline Delays	211
6.3.1	Branch Delays	212
6.3.2	Structural Hazards	225
6.3.3	Data Dependencies	227

6.4	Interrupts	231
6.5	Superscalar Pipelines	234
6.6	Pipelined Processor Memory Demand	237
	REFERENCES	239
	EXERCISES	239
7	INPUT/OUTPUT	241
7.0	Introduction	241
7.1	I/O System Architecture	241
7.2	I/O Device Requirements	249
7.3	Buses and Controllers	252
7.3.1	Bus Design Examples	258
7.4	Serial Communication	258
7.5	Clock Synchronization	263
7.6	Queuing Theory	265
7.6.1	Open-System Queuing Model	265
7.6.2	Closed-System Queuing Model	271
	REFERENCES	273
	EXERCISES	274
8	EXTENDED INSTRUCTION SET ARCHITECTURES	275
8.0	Introduction	275
8.1	Operating System Kernel Support	276
8.2	Virtual-Memory Support	283
8.3	Interrupt Support	290
8.4	Input/Output Support	293
8.5	Cache Support	295
8.6	Multiprocessor Support	298
8.7	Other User ISA Enhancements, MMX	302
8.7.1	Other Intel ISA Support	307
	REFERENCES	308
	EXERCISES	308
	<i>Index</i>	310
	<i>Trademarks</i>	318

PREFACE

This textbook is intended for use in a one-semester, upper-division undergraduate course. It is expected that the students will have had courses in logic design with state machines and assembly language programming. A course in data structures as well as basic operating systems would be helpful.

There are two major themes running through this book. The first theme is an overview of the major ideas and design philosophies of computer architecture and implementation with some insight into their historical development. We are inclined to forget the work of our predecessors in computer design; the few pages devoted to this topic may help to rectify this neglect.

The second theme is the introduction and use of analytic modeling of computer performance. I believe that engineering students should take an engineering approach to the study of computer architecture and implementation. Various design trade-off issues will be examined with analytical models. What is an engineering approach? Consider the following two quotations.

“What distinguished the engineer from the technician is largely the ability to formulate and carry out the detailed calculations of forces and deflections, concentrations and flows, voltages and currents, that are required to test a proposed design on paper with regard to failure criteria. The ability to calculate is the ability to predict the performance of a design before it is built and tested.” (Petroske 1996).

“... it is much easier to have some vague notion about any subject, no matter what, than to arrive at the real truth about a single question, however simple that may be.” (Descartes 1629).

In addition to the two major themes, this book will introduce students to a new vocabulary. Whenever a new topic is approached, it is inevitable that one learns a new vocabulary. Therefore, whenever a new word or term is introduced, this book attempts to provide a definition along with common synonyms. We recommend that students obtain a copy of a computer dictionary to support their studies. Reference can also be made to an on-line dictionary of computer terms at (<http://wombat.doc.ic.ac.uk/foldoc/index.html>).

One of the problems in the computer field is the lack of a standard set of terms. For example, the Intel x86 architecture uses the term linear address whereas I prefer the

term virtual address, as used with the IBM S370 and S390. The reason for the preference is that the IBM S370 predates the virtual-memory version of the Intel x86 and established precedence. Thus, in the text, wherever possible, the term with precedence is used.

The von Neumann architecture is used in Chapter 1 to introduce the basic principles of computer architecture. Students should know what von Neumann proposed with this architecture and what its limitations were. These limitations were overcome in subsequent architectures while the underlying design remained unchanged.

Two architecture families will be extensively used for illustrations: The IBM S360/370/390 and the Intel x86 up to the Pentium Pro with MMX extensions. These two families represent the two most widely used processors today. For mainframe applications, the IBM family is preeminent in the world. The Intel family is found in approximately 90% of all personal computers. These two families are the most likely to be encountered by students after they finish their studies.

The sequence of presentation in this book differs from that found in other texts. It starts with a computer overview in Chapter 1 that covers the von Neumann architecture and some of its precursors. Chapter 2 introduces analytical performance modeling, learning curves, and other related evaluation techniques. The modeling technique will be used throughout the balance of the book.

Chapter 3 describes the issues in user instruction set design. The instruction set architecture taxonomy of addresses, data types, and operations is introduced. This chapter also provides a basic treatment of computer arithmetic and the IEEE floating-point standard.

Because a memory-centric view of a computer is taken, the text turns to memory implementation in Chapter 4. I believe that students should be exposed to memory systems before processors, as the process state is held memory and is the focus of how problems are solved. Further, I approach the memory system by first looking at virtual memory that provides the functionality required of large programs and large data sets.

Performance follows functionality, from a designer's point of view; thus caches are covered after virtual memory. The gap between the cache and the disk is then covered by describing interleaved memory and DRAM organizations. Performance models for hierarchical memories are used throughout the chapter to illustrate design trade-offs.

Following the chapter on memory, the issues of processor implementation are addressed. First, hardwired and microprogrammed implementations are covered in Chapter 5. Then Chapter 6 addresses the ideas and issues of pipelining. The techniques of performance models are interwoven with the functional descriptions of processor implementation.

Chapter 7 covers input/output. In this chapter we look at requirements for various workloads and examine how these requirements are satisfied in contemporary systems. Bus design and serial communications are discussed along with clock recovery and clock synchronization. A brief introduction to queuing theory is included in this chapter.

The concluding chapter, Chapter 8, covers extensions to the user instruction set architecture that support operating systems, virtual memory, caches, and multiprocessors.

x Preface

This chapter concludes with a description of the MMX extensions to the Intel Pentium Pro.

A final goal of this book is to enable students to read the current literature on computers with understanding and appreciation of the ideas presented. This is accomplished by exposing students to the basic principles of architecture and implementation, and by careful use of the nonstandard terminology of this field.

REFERENCES

- Petroski, H. (1996). *Invention by Design*, Harvard U. Press, Cambridge, MA, pp. 89–90.
Descartes, R. (1692). *Rules for the Direction of the Mind*.

TABLE 3.2 INSTRUCTION FORMAT BITS

Architecture Type	Bits
Three address, memory to memory	$P + 3A$
Two address, memory to memory	$P + 2A$
Three address, register to register	$P + 3R$
Load/store	$P + A + R$
Two address, register to register	$P + 2R$
Load/store	$P + A + R$
One address, accumulator	$P + A$
Zero address, stack	P
Push/pop	$P + A$
Multiple accumulator	$P + A + R$

- Multiple accumulator:

Op-Code	A	R
---------	---	---

$$R(R) \leftarrow (R(R)) \text{ op } (M(A)).$$

The multiple accumulator architecture selects a register from the register file to be an accumulator. Because both a memory and a register address are available, special load and store instructions are not required.

The various instruction compositions (and local storage) are never found as “pure” architectures except in RISC designs. CISC ISAs are composed of several of these design options. The Intel Pentium supports load/store register file for integers, accumulator for integers, and an evaluation stack for floating point. A combination of the accumulator, register-to-register, and multiple accumulator architectures are used with the IBM S360.¹⁷

Bit Budget

Observe that each of these register organizations has differing numbers of bits in their instruction formats. A major consideration in ISA design in the early decades of the computer industry was the minimization of the bits required to represent a program. The number of bits is known as the bit budget.¹⁸ To illustrate how the bit budget is computed, assume that an op-code has P bits, a memory address has A bits, and a register address has R bits. Table 3.2 tabulates the number of bits required for each instruction format.

¹⁷ The designation S360 is used in this text to identify all members of this ISA family: S360, S370, S390. The specific designation will be used when the feature being discussed applies to one of the family members.

¹⁸ Bit budget may be measured in bits, bytes, words, or instructions. Any consistent metric will suffice.

system. Following the technique of the example in Subsection 3.2.3.1 we find that 3 bits are saved in the exponent for equal range.

Comment

From a bits-lost-or-saved point of view, base 256 seems to be a poor choice: lose 4 bits in the fraction, save 3 bits in the exponent.

Logical Data Types

Many computers define logical data types, also called Booleans, that are a vector of bits allocated to a byte, half-word, word, or double-word primary data type. These bit vectors are used to record binary events or states. Logical data types are used primarily for housekeeping tasks within a program and are rarely found as the input or the output of a program.

The S360 allocates logical values to any of the data types, bytes, half words, words, and double words. The Pentium Pro can allocate a bit field of any length, as shown in Figure 3.11. One address specifies the start of the field, and a length parameter specifies the number of bits in the field.

3.2.3.2 Program Sequencing Data Types

A number of architected data types are used for controlling the sequence of the executing programs. These data types are the program counter and the indicator bits.

Program Counter

With modern computers, a program is stored in sequential memory locations and is accessed by the program counter. The program counter data type is an unsigned integer having a length equal, in many architectures, to one of the primary data types of the ISA. It is convenient to have the program counter data type the same length as a user data type, so it can be stored in memory with a context switch.²⁹ The Pentium Pro program counter is a 32-bit register called the extended instruction pointer (EIP) register.

The IBM S360 takes a different approach. The program counter (called the instruction address) is allocated in the 64-bit program status word (PSW), a double word that can be conveniently stored. The S360 and S370 program counters are 24 bits, whereas the S390 has a 31-bit program counter.

Branches require an EA that is loaded into the program counter when a branch is taken. There are three common forms of branch EAs that become the new program counter value. The displacement is the displacement field of the branch instruction:

Direct	$PC \leftarrow (\text{displacement})$
PC relative	$PC \leftarrow (PC) + (\text{displacement})$
Base relative	$PC \leftarrow (\text{base}) + (\text{displacement})$

²⁹ Some single-chip microcomputers do not follow this convention. The length of the program counter is set to address the on-chip program ROM. A special hardware stack is provided to save the program counter for the return address when calling a subroutine or when responding to an interrupt.

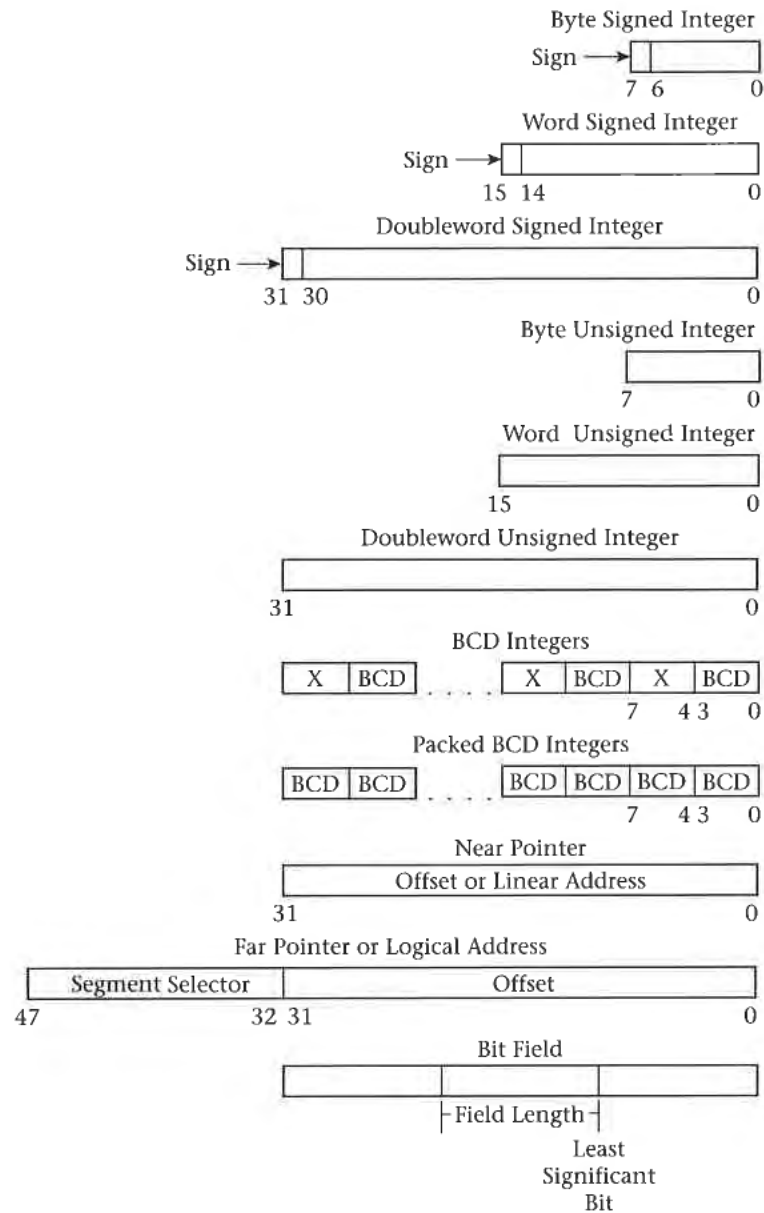


Figure 3.11 Pentium Pro data types (courtesy of Intel Corporation)

EXAMPLE

What happens when a displacement is added to the PC or a base value and there is a carry out?

Solution

There are two solutions to this design problem. First, the carry can be detected and an interrupt signaled, indicating an error. Second, the effective address wraps around and the branch is taken to a low-order address.

Comments

A designer must ensure that all such unusual situations do not result in unpredictable performance.

Indicator Bits

As data types, indicator bits are simply bits. The interesting characteristics of this data type are the methods by which they are set/reset. In Subsection 3.2.2 the use of indicator bits by branch instructions is discussed. Remember that the indicator bits permit a condition to be carried forward to a subsequent instruction that will test and branch based on the condition. Indicator bits are set/reset or recorded by two methods:

- as a side effect of an instruction
- explicitly as the primary result of an instruction

An example of side effect recording is as follows. For all arithmetic operations, the output bus of the ALU is tested for zero; if zero, the indicator is set to 1; otherwise it is reset to 0. Explicit instructions can also record a result in an indicator. For example, a Pentium Pro instruction performs the following:

CMP – compare two operands. Compare the contents of two registers and set the status flag according to the results.

With most early computers, such as the von Neumann ISA, the branch instructions were of the test-and-branch type and did not use indicator bits to carry forward condition information.³⁰ The first use of indicator bits known to the author is found in the IBM 704, first installed in 1955. With the IBM 704, an input byte is checked by hardware and its parity recorded in a P bit. This bit holds the parity condition until there is another input operation that sets/resets the indicator. Thus the indicator must be used before it is changed by a following instruction. All other branches in this computer were of the test-and-branch type that tests the condition of various registers.

The use of indicator bits, as we know them today, can be traced to the IBM Stretch installed in 1961. With this machine, 64 indicator bits are grouped into an architected double word that, in addition to being tested by branch instructions, could be stored and reloaded for context switching. Seven of the bits recorded the result of an integer ALU operation as side effects: negative, zero, positive, negative sign, low, equal, and high. Ten of the bits recorded the floating-point ALU results and exceptions.

IBM S360 Condition Codes. The IBM S360 carried forward the architected indicator bits idea from the IBM Stretch in a restricted fashion. The condition code is allocated to the architected PSW along with other status information. The designers of the S360 believed that the indicators of the Stretch could be encoded, thereby

³⁰ The von Neumann ISA could branch on a previously computed value in the accumulator, ≥ 0 . Likewise, the Bell Model V relay calculator could branch on the sign of a previously computed value.

TABLE 3.13 IBM S360 CONDITION CODES

Instruction	Condition Code Value			
	00	01	10	11
ADD logical	zero, no carry	not zero, no carry	zero, carry	—
SUBTRACT logical	=	not zero, no carry	zero, carry	not zero, carry
Test under mask	zero	mixed	—	ones
ADD	zero	<zero	>zero	overflow
Move long	count equal	count low	count high	dest. overlap
Compare floating point	equal	low	high	—
ADD decimal	zero	<zero	>zero	overflow
Store clock	set	not set	error	not operational

saving bits in the PSW and making room for the program counter and other control information.

Two bits are used to encode four conditions; encoded indicators are called condition codes. Further complicating the design, the conditions are recorded as side effects and are context dependent. That is, the meaning of the encoded value depends on the instruction that records the condition. A small sample of the condition coding is shown in Table 3.13.

The branch instructions have a 2-bit field that is compared with the 2-bit condition code. For example, if after performing an ADD one wants to test the result for zero, the 2 bits in the instruction would be 00. If a branch on overflow is desired, the bits would be 11.

The system of the IBM S360 opens many opportunities for programming errors. Say, for example, an ADD is performed; then the clock is stored. The information in the condition code following the add has been destroyed. A branch on 00 would now report that the clock is set, not that the result of the previous add is not zero.

Intel Architecture Family Indicator Bits. Intel literature refers to indicators as flags, the term used in this discussion. The first member of the x86 family, the i8008, has four implemented flags that are set/reset as side effects to other operations: carry, zero, positive, parity. Because the flags are implemented, their values cannot be stored and retrieved for context switching.

As the evolution of the x86 family progressed to the i8080 and the i8086, the i286 was the first member with architected flags. With the i286, 16 flags are allocated in a register that can be stored and retrieved for context switching. However, only 12 flags were assigned, leaving 4 bits for expansion.

The flags of the Intel Pentium Pro are shown in Figure 3.12. The flags are allocated to a 32-bit word. Note there are three classes of flags: status, control, and system. The auxiliary carry flag is used for performing a decimal adjust after a decimal addition. The alignment check flag is of interest because the x86 memory is byte addressable. If, for example, a word (4 bytes) is addressed and the address is not

- 3.20 The Atanasoff Berry Computer, described in Chapter 1, uses Gaussian elimination to solve linear algebraic equations; see Figure 1.9. Show how the ABC solves for x and y by using binary data types; $3x + 2y = 12$, $4x + y = 11$.
- 3.21 Can the two steps of the x86 decimal-adjust operation be combined into one step? If so, describe the operation. If not, explain why not.
- 3.22 The text states in Section 3.3 that one advantage of a fixed-length instruction is that instructions will not cross page boundaries. Explain why this is so.
- 3.23 Consult Intel documentation on any member of the x86 family and explain how signed decimal arithmetic is performed.
- 3.24 The combinations of fixed-length/variable-length instructions and op-codes are discussed in Subsection 3.2.4. Why is the combination of fixed-length instructions and op-codes not included in the list of design options?
- 3.25 The program counter of the VAX is allocated to register 15 of the register file. Discuss the advantages and disadvantages of this design.
- 3.26 Consult the documentation for the IBM S390 and construct the equivalent of Table 3.13 IBM S360 op-code classes.
- 3.27 The example in Subsection 3.2.3 shows that base 16 requires 2 fewer bits than base 2 for the same range. The equality was solved by assuming one base 16 exponent bit. Show another solution of the equality that makes no such assumption and shows that the result is general.
- 3.28 The average length of an 8086 instruction was given as 2.25 bytes in Section 3.3. This average was found by running programs in 1989. At this time, do you think the number of bytes per Pentium Pro instructions is less than or greater than 2.25? Explain your answer and give an estimate that you would use for modeling purposes.
- 3.29 Provide a rationale for the round-to-nearest-domain function of the IEEE floating-point standard for the case in which $R = 1$ and $S = 0$.
- 3.30 Figure 3.9 shows three classes of interrupt handling methods. However, the text suggests that there are four: the combinations of between instructions or within and instruction and continuation and restart. Explain how these four methods are accommodated in Figure 3.9.
- 3.31 Generalize the rules for making the correction to biased exponents after an addition or subtraction.
- 3.32 Are the Intel integer data types one of the six data types described in Table 3.6? Explain your answer.

that has a control bit (bit 15) set to supervisor mode.⁷³ The user PSW is saved in memory location 32. After the supervisor call interrupt has been serviced, the privileged load PSW (LPSW) instruction restores the old PSW and restarts the interrupted program.

Comment

If the PR instruction is executed and the processor is not in the supervisor mode, an illegal op-code interrupt will be generated when the instruction is decoded.

The IBM S390 introduces semiprivileged instructions that can be executed in the user mode when certain authority requirements are met. Likewise, the Intel Pentium Pro has four levels of privilege. Level 0 is the operating system kernel, levels 1 and 2 are the operating system services (device drivers, etc.), and level 3 is the user or application level.

In this chapter, we discuss the privileged instructions, their operations, data types, and addresses. Two classes of extended user instructions are also discussed in Section 8.7: multiprocessor support and MMX enhancements to the Intel Pentium.

8.1 OPERATING SYSTEM KERNEL SUPPORT

A subset of privileged instructions, operating system kernel support, is discussed in this section. There are some operating system instructions that are similar to user instructions except that they use different registers. Processors may have a set of registers that are addressed only by the privileged instructions. Thus there are, in essence, other processors embedded in the user processor. However, there are common resources such as caches, ALUs, and implemented registers used by all instructions.

The processor state consists of the system data types that are collections of bits, fields of bits, and words allocated to registers that are used as semifixed indicators of some state of the process and processor. These registers hold indicator bits that define at any time the current operating mode and conditions of the processor. The processor state registers usually include the indicator bits, discussed in Chapter 3 in the context of branching. In addition to the state register data types, the memory addresses associated with these data types are described along with the operations that can be performed on the data types. The processor state registers for the IBM S370 and the Intel Pentium Pro are

IBM S370 (IBM 1976):

PSW

Control registers

Intel Pentium Pro (Intel 1996):

Flag register

Control registers

Instruction pointer register

⁷³ IBM calls this bit *P* for problem state bit.

The PSW, flag register, and the instruction pointer register hold bits that may be changed dynamically during program execution. Control registers hold bits that are not normally modified during program execution. These bits establish the mode of operation and are swapped to/from memory because of a context switch or, with some processors, a subroutine call.

The data types described in this chapter have many bits and fields allocated to computer words. Students need not memorize these allocations, as detailed knowledge can always be obtained from the manufacturers' literature. However, students should understand why the data types exist and have a general understanding of the function and the purpose of the data types.

The operating system kernel data types exist to control the system, not produce results. Thus they never leave the system as outputs. The data types are transferred to dedicated registers in the processor, where they perform their intended function. These data types may be modified by privileged instructions during program execution or by side effects to user instructions.

IBM S360, S370, and S390

The IBM S360 family of computers was designed from the beginning to be controlled by an operating system. Thus a number of hardware and software concepts were introduced that facilitate the design and use of an operating system. We look at the data types, addresses, and operations that support the operating system kernel.

Data Types. Figure 8.1 shows the PSWs for the IBM S360 basic control (BC) mode and the S370 extended control (EC) Mode. The S370 extended the S360 with virtual memory requiring additions to the S360 PSW. Note that the program counter, called the instruction address, is found in the PSW as is the 2 bit conditing code (CC). Bits 0–7 set the control of the interrupts.

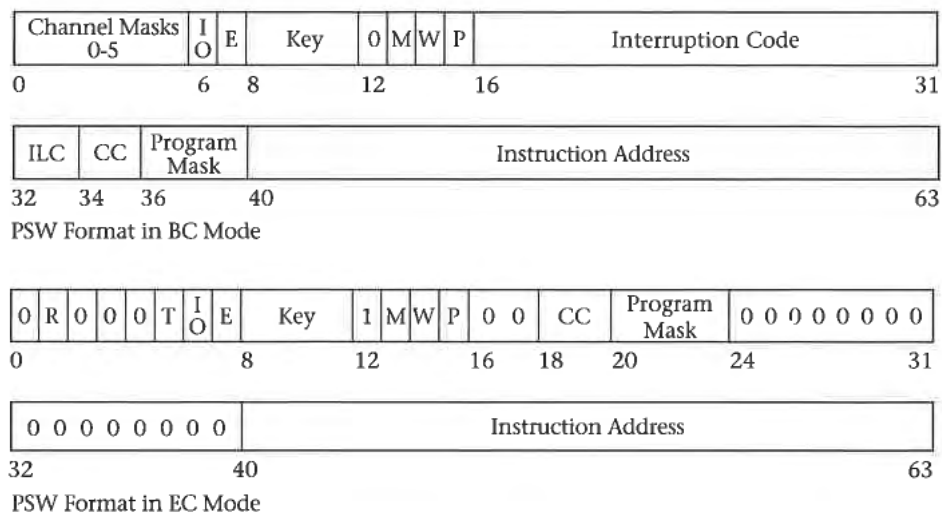


Figure 8.1 IBM S360, S370 PSWs (courtesy International Business Machines Corporation)

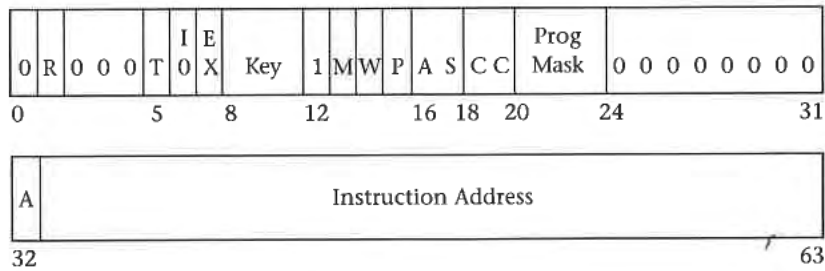


Figure 8.2 IBM S390 PSW (courtesy International Business Machines Corporation)

A 4-bit key provides protection for 2048-byte blocks in memory.⁷⁴ Each block in memory has an associated 7-bit key, called key in storage. For CPU writes, the 4-bit key in the PSW must match the corresponding 4-bit key in storage. CPU reads are permitted if the key matches or if bit 4 in the key in storage is zero. This key mechanism provides protection from unauthorized reads and writes. The key in the PSW is modified by a privileged instruction.

The 4-bit program mask augments the condition code for fixed-point overflow, decimal overflow, floating-point underflow, and floating-point significance when the result fraction of a floating-point operation is zero. An interrupt is generated when any of the mask bits are set to 1. The operating system kernel then determines the source of the interrupt by examination of the condition code, key, and/or the interrupt code.

The S390 PSW, EC mode, reallocates all the bits except the instruction address, key, and a few other bits, as shown in Figure 8.2. The major difference from the S370 is that the instruction address has been increased to 31 bits from 24. Reference should be made to an operations manual for a full description of the bits.

In addition to the PSW data type, the IBM S370 has a 16-word 32-bit control register (CR) file.⁷⁵ One or more bits in these registers is assigned to functions needing control or specification. Some of the bits are assigned to virtual-memory address translation and others to multiprocessing (for example, the time of day clock mask). Other bits control the recording of program events, such as successful branches. Finally, bits control how machine checks are handled: logout or not logout. Table 8.1 shows the general functions allocated to each of the CR words. Refer to any detailed architectural description of the IBM S370 to obtain the exact allocations.

Addresses. The contents of the PSW and the CRs are moved to/from memory by special instructions. Thus a source and a destination address are required. The PSW and CRs are addressed by implication in the instruction op-code, and memory is addressed by a computed effective address.

The S370 provides a region of real memory that is directly accessed – i.e., the addresses are not mapped by the virtual-memory system. This region of memory, 0–1FC, is used to store PSWs, CRs, and machine-check information. Space is also provided to

⁷⁴ A block may or may not be a virtual-memory page as the S370 can have 4096-byte pages.

⁷⁵ The S360 did not have CRs; they were added with the S370 and their use expanded with the S390.

TABLE 8.1 IBM S370 CR ALLOCATIONS

Word	Primary Use
0	Multiprocessing, dynamic address translation, timer
1	Dynamic address translation
2	Channel masks
3	Unassigned
4	Unassigned
5	Unassigned
6	Unassigned
7	Unassigned
8	Monitor
9	Program-event recording
10	Program-event recording
11	Program-event recording
12	Unassigned
13	Unassigned
14	Machine-check handling
15	Machine-check handling

store floating-point and general register values. The reason this region is direct mapped is to ensure that its contents are never paged out to the disk; the contents are always directly accessible by the processor.

Operations. Before the operations are described, it is instructive to examine the six instruction formats of the IBM S370 shown in Figure 8.3. The instructions can be 2, 4, or 6 bytes long. A wide variety of addressing options are provided. The formats associated with each of the instructions of interest are indicated in the text by their two-character format codes, such as RR for register to register.

The IBM S360 has 11 privileged instructions: 5 for the kernel (plus 1 nonprivileged instruction), 4 for I/O, and 2 for multiprocessing. The IBM S360 set of instructions is a smaller set than the 26 of the IBM S370 and the 55 privileged and semiprivileged instructions of the IBM S390. For instructional purposes, the simple S360 set is described.

Table 8.2 shows the instructions that are used to modify the status of the CPU by loading and storing the bits in the PSW. Note that the storage key can be loaded or stored but the system mask and the program mask can only be loaded into the PSW. The diagnose instruction invokes hardware diagnostics that are unique to the processor model.

Intel Pentium Pro

The Intel Pentium Pro evolved from the Intel 8008, which was not designed to be under the control of an operating system. Thus some of the features have the appearance of being added on to a simpler architecture. The operating system has four levels of privilege with processor state data types, addresses, and operations for the support of the operating system kernel.

A processor accesses the semaphore and determines if it can use the datum or if the datum is in use by another process. Reading the semaphore must be done in such a way that any other processor cannot attempt to access the semaphore while this is going on. In other words, there must be exclusive access to the semaphore in real memory. A basic design problem in achieving this end is that the instruction must be able to read-modify-write to the memory location holding the semaphore; two data memory cycles are required.

One way of ensuring exclusive access to the semaphore is by means of an atomic instruction. An atomic instruction is an instruction that performs an operation that cannot be interrupted or terminated by another processor and the semaphore cannot be accessed by another processor during the execution of the instruction.

Two distinctly different techniques are used to achieve atomic synchronization instructions with the IBM S360 and the Intel x86 architecture (before the Pentium Pro). These are discussed in the two subsequent subsections.

IBM S360

One type of synchronization instruction is test and set, first used on the IBM S360 and included in the ISA of the S370 and S390 (Falkoff et al. 1964). This instruction is not privileged, is atomic, and performs the following steps⁸⁰:

1. As a side effect, it locks out the memory bus from other users.
2. It copies the state of *S* into the CC bits of the PSW.
3. It performs a write that sets *S* to 1. If *S* is already 1, its value remains 1. (Collectively, steps 1, 2, and 3 are the test and set instruction. After this instruction is executed, the lock on the memory bus is released.)
4. The processor then tests the condition code with a branch instruction to determine if access to the datum has been obtained. If $CC = 0$, access is obtained, go to step 5. If $CC = 1$, access is not obtained; the processor tries again by returning to step 1.
5. The processor uses the protected datum, which may include modifying the datum, and goes to Step 6.
6. It executes an atomic store instruction that clears *S* to 0; thereby releasing the datum to another user, then continues.

Data Types. There are two data types associated with synchronization. The first is the semaphore, which is a byte rather than a single bit. This approach wastes 7 bits but simplifies the design as this architecture addresses to the byte; all the bits of the byte are either 1 or 0.

The second data type is the 2-bit CC field of the PSW shown in Figures 8.1 and 8.2. These bits are set to 00 for a semaphore of 0 and 01 for a semaphore of 1.

⁸⁰ IBM does not use the word atomic. Instead they use the phrase “appears to be an interlocked-update reference as observed by other CPUs.” An I/O channel is not considered to be a CPU and interlocks are not provided.

TABLE 8.18 IBM S360 PROCESSOR SYNCHRONIZATION INSTRUCTIONS

Name	Mnemonic (Format)	Function
Test and set	TS (SI)	The byte at address ($D_1 + (B_1)$) is copied into the condition code and 1 is written into all bits of the addressed byte
Move	MVI (SI)	Moves the immediate byte of all 0s in bits 8–15 into the byte at address ($D_1 + (B_1)$)

Addresses. The effective address of the semaphore byte is base plus displacement. The other address for test and set, the CC, is implied in the op-code. The atomic store instruction uses an immediate operand of all zeros for the source. The destination address is the effective address of the semaphore byte.

Operations. Synchronization is provided by two atomic instructions, shown in Table 8.18. There are a number of complications in implementing the test and set instruction. For example, during a test and set instruction, the bus and memory are locked out from other users. As this instruction can take many clocks, the bus utilization can be adversely affected. An extreme case for the S370 and S390 would be that the first time the semaphore is accessed, it may be on the disk in virtual-address space, requiring a page fault, all the while locking out other processors from the memory system.

Other Multiprocessor Instructions. The IBM S360 provides a direct path for processors to communicate without using the system bus. Each processor has eight direct-in lines, eight direct-out lines, and eight direct-signal lines (bits 8–15 of the SI format instruction). Two instructions, shown in Table 8.19, use these lines for communication between other processors in the system.

There is a strobe signal with model-specific timing specifications that are not pertinent to the discussion of ISA.

TABLE 8.19 IBM S360 INTERPROCESSOR COMMUNICATION INSTRUCTIONS

Name	Mnemonic (Format)	Function
Read direct	RDD (SI)	Bits 8–15 of the instruction are placed on the signal-out lines. The byte on the direct-in lines is written into the byte at memory address ($D_1 + (B_1)$)
Write direct	WRD (SI)	Bits 8–15 of the instruction are placed on the signal-out lines. The byte at memory address ($D_1 + (B_1)$) is placed on the direct-out lines

TABLE 8.22 INTEL PENTIUM PRO PERFORMANCE MONITOR EVENTS

Unit Monitored	Event Description
Data cache	• All memory references • Weighted number of cycles while a data cache miss is outstanding
Instruction fetch unit	• Number of instruction fetches • Number of instruction TLB misses • Number of instruction fetch misses
L2 cache	• Number of L2 instruction fetches • Number of L2 data loads • Number of L2 data stores
Floating-point unit	• Number of computational floating-point operations retired • Number of computational floating-point operations executed

Note that these performance monitoring facilities are similar to those found with the IBM S370. Table 8.1 shows that words 9, 10, and 11 are allocated to recording program events.

REFERENCES

- Dijkstra, E. W. (1968). "Co-Operating Sequential Processes," in *Programming Languages*, F. deGenuys, ed., Academic, New York.
- Falkoff, J. W., Iverson, K. E., and Sussenguth, E. H. (1964). "A Formal Description of System /360," *IBM Systems Journal*, **3**:3.
- Flynn, M. J. (1966). "Very High-Speed Computing Systems," *Proceedings of the IEEE*, **54**, pp. 1901-1990.
- IBM (1970). "IBM S/360 Principles of Operation," order GA22-6821-8.
- IBM (1976). "IBM System/370 Principles of Operation," order GA22-7000-5.
- IBM (1997). "Enterprise System Architecture/390 Principles of Operation," order SA- 7201-04.
- Intel Corporation (1993). "Pentium Processor User's Manual," Vol. 3, order 241-430-001.
- Intel Corporation (1996). "Pentium Pro Family Developer's Manual," Vols. 1-3, order 000901-001.
- Intel Corporation (1997). "Intel Architecture MMX™ Technology, Programmer's Reference Manual," order 243007-003.
- May, C., Silha, E., Simpson, R., and Warren, H., eds. (1994). *The PowerPC Architecture*, Morgan Kaufmann, San Francisco, CA.
- Peleg, A. and Weiser, U. (1996). "MMX Technology Extension to the Intel Architecture," *IEEE Micro*, **16**:4, pp. 42-50.

EXERCISES

- 8.1 Describe the IBM S360/370 channel ISA in terms of data types, addresses, and operations.
- 8.2 What instruction could be used with the IBM S360 to determine if the condition code permits access to a datum that is protected with a semaphore?

- 8.3 Consult the Intel Pentium Pro documentation on performance monitoring and determine how the miss rate of the instruction cache and the data cache can be determined.
- 8.4 The paper "PicoJava: A Direct Execution Engine for Java Bytecode" by McGhan and O'Connor, found in *Computer*, October 1998, describes a processor for executing Java programs. Write an essay describing the salient architecture and implementation features of this processor in 500 words or less.
- 8.5 Describe the meaning of the circled numbers 1, 2, and 3, of Figure 8.6.
- 8.6 Section 4.6 describes four segment/page configurations that are used with the IBM S370. For each of these configurations, list the number of bits that are added in the two adders and the bits concatenated in the real-address register of Figure 8.6. Recall that the S370 has a 24-bit effective or logical address.
- 8.7 What is the length of the virtual address of the Pentium Pro?
- 8.8 Figure 8.8 shows the Pentium Pro segmented virtual-address system for 4K-byte pages. The system can also support 1M-byte pages. Show how the linear address and tables are reconfigured for 1M-byte pages.
- 8.9 Explain why two adders are used for the S370 virtual-address translation, Figure 8.6, whereas only one adder is used for the Pentium Pro, Figure 8.8.
- 8.10 Consider the synchronization instructions discussed in Section 8.6. What happens when two or more requests to access the semaphore arrive on exactly the same clock?
- 8.11 Derive the model for MIOPS used in the example in Section 8.7.
- 8.12 What is done with the two 32-bit results computed in the multiply and add instruction shown in Figure 8.19? In other words, how can these results be used as inputs to subsequent instructions? Note that the source operands are 16 bits.





Computer architecture and implementation

Author: [Harvey G. Cragon](#)

Summary: "This textbook provides a clear and concise introduction to computer architecture and implementation. Two important themes are interwoven throughout the book. The first is an overview of the major concepts and design..."

[Show more](#) ▾

 **Print Book**, English, 2000

Edition: [View all formats and editions](#)

Publisher: Cambridge University Press, Cambridge, 2000

Physical Description:	x, 317 pages : illustrations ; 26 cm										
ISBN:	9780521651684, 9780521657051, 0521651689, 0521657059										
OCLC Number / Unique Identifier:	41445657										
Subjects:	ARCHITECTURE (COMPUTERS) COMPUTER DESIGN COMPUTERS Computer architecture INPUT/OUTPUT ROUTINES... Show more ▾										
Contents:	<table><tr><td>Preface</td><td>viii</td></tr><tr><td>Computer Overview</td><td>1 (25)</td></tr><tr><td>Introduction</td><td>1 (1)</td></tr><tr><td>von Neumann Model</td><td>2 (3)</td></tr><tr><td>The von Neumann Architecture</td><td>5 (8)</td></tr></table> Show more ▾	Preface	viii	Computer Overview	1 (25)	Introduction	1 (1)	von Neumann Model	2 (3)	The von Neumann Architecture	5 (8)
Preface	viii										
Computer Overview	1 (25)										
Introduction	1 (1)										
von Neumann Model	2 (3)										
The von Neumann Architecture	5 (8)										
More Information:	Table of contents Publisher description										

[Show less information](#) ▴

Find a Copy at a Library

Filter by: [Any format](#) ▾ [Any edition](#) ▾ [Distance within 200+ mi](#) ▾

 5 editions in 310 libraries

Screen capture by J L Mullins from *WorldCat* on October 25, 2022

Regional or National



[Library of Congress](#)

📍 2.2 miles from your current location.

101 Independence Avenue, SE
Washington, DC 20540
United States

[Get Directions](#) 


Favorite Library

[Borrow](#) 

LIBRARY

CATALOG

←

<

1 of 1

>

BOOK

Computer architecture and implementation

Full Record

MARC Tags

Personal name

Cragon, Harvey G.

Main title

Computer architecture and implementation / Harvey G. Cragon.

Published/Created

Cambridge ; New York : Cambridge University Press, 2000.

Request this Item

 LC Find It

Links

>

Links

Table of contents <http://www.loc.gov/catdir/toc/cam024/99016243.html>

Publisher description <http://www.loc.gov/catdir/description/cam029/99016243.html>

More Information

>

LCCN Permalink

<https://lccn.loc.gov/99016243>

Description

x, 317 p. : ill. ; 26 cm.

ISBN	0521651689 (hard) 0521657059 (pbk.)
LC classification	QA76.9.A73 C74 2000
LC Subjects	Computer architecture.
Browse by shelf order	QA76.9.A73
Notes	Includes bibliographical references and index.
LCCN	99016243
Dewey class no.	004.2/2
Type of material	Book

Item Availability

>

CALL NUMBER	QA76.9.A73 C74 2000 LANDOVR Copy 1
Request in	Jefferson or Adams Building Reading Rooms - STORED OFFSITE
Status	Not Charged

CALL NUMBER	QA76.9.A73 C74 2000 FT MEADE Copy 2
Request in	Jefferson or Adams Building Reading Rooms - STORED OFFSITE
Status	Not Charged

LIBRARY | CATALOG

◀ 1 of 1 ▶

BOOK

Computer architecture and implementation

[Full Record](#)

[MARC Tags](#)

000	01176cam a22003014a 4500
001	4520830
005	20020905120143.0
008	990525s2000 enka b 001 0 eng
035	— 9 (DLC) 99016243
906	— a 7 b cbc c orignew d 1 e ocip f 19 g y-gencatlg
925	0_ a acquire b 2 shelf copies x policy default
955	— a pc03 to ja00 05-25-99; jf03 to Subj. 05-25-99; jf11 to sl 05-26-99; jf12 05-26-99 to ddc; CIP ver. jg15 05-03-00; ja15 to BCCD 07-13-00
010	— a 99016243
020	— a 0521651689 (hard)
020	— a 0521657059 (pbk.)
040	— a DLC c DLC d DLC
042	— a pcc
050	00 a QA76.9.A73 b C74 2000
082	00 a 004.2/2 2 21
100	1_ a Cragon, Harvey G.
245	10 a Computer architecture and implementation / c Harvey G. Cragon.
260	— a Cambridge ; a New York : b Cambridge University Press, c 2000.
300	— a x, 317 p. : b ill. ; c 26 cm.
504	— a Includes bibliographical references and index.
650	_0 a Computer architecture.
856	41 3 Table of contents u http://www.loc.gov/catdir/toc/cam024/99016243.html

85642|3 Publisher description |u http://www.loc.gov/catdir/description/cam029/99016243.html

984—|a gsl

Request this Item

 LC Find It

Item Availability

>

CALL NUMBERQA76.9.A73 C74 2000 LANDOVR
Copy 1

Request inJefferson or Adams Building Reading Rooms - STORED OFFSITE

StatusNot Charged

CALL NUMBERQA76.9.A73 C74 2000 FT MEADE
Copy 2

Request inJefferson or Adams Building Reading Rooms - STORED OFFSITE

StatusNot Charged

PROCEEDINGS

29

COMPSAC^Φ05

**The Twenty-Ninth Annual International
Computer Software
& Applications Conference**

26-28 July 2005

Edinburgh, Scotland

EDINBURGH




IEEE
COMPUTER
SOCIETY

 **IEEE**



NORTH CAROLINA STATE UNIVERSITY LIBRARIES



S02538439 Y

Proceedings of the

**29th Annual International Computer
Software and Applications Conference
(COMPSAC 2005)**

July 26–28, 2005

Edinburgh, Scotland

Sponsored by

IEEE Computer Society

Supported by

Motorola



IEEE

**COMPUTER
SOCIETY**

<http://computer.org>

Los Alamitos, California

Washington • Brussels • Tokyo

Copyright © 2005 by The Institute of Electrical and Electronics Engineers, Inc.
All rights reserved.

Copyright and Reprint Permissions: Abstracting is permitted with credit to the source. Libraries may photocopy beyond the limits of US copyright law, for private use of patrons, those articles in this volume that carry a code at the bottom of the first page, provided that the per-copy fee indicated in the code is paid through the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923.

Other copying, reprint, or republication requests should be addressed to: IEEE Copyrights Manager, IEEE Service Center, 445 Hoes Lane, P.O. Box 133, Piscataway, NJ 08855-1331.

The papers in this book comprise the proceedings of the meeting mentioned on the cover and title page. They reflect the authors' opinions and, in the interests of timely dissemination, are published as presented and without change. Their inclusion in this publication does not necessarily constitute endorsement by the editors, the IEEE Computer Society, or the Institute of Electrical and Electronics Engineers, Inc.

IEEE Computer Society Order Number P2413

ISBN 0-7695-2413-3

ISSN 0730-3157

Additional copies may be ordered from:

IEEE Computer Society
Customer Service Center
10662 Los Vaqueros Circle
P.O. Box 3014
Los Alamitos, CA 90720-1314
Tel: +1 800 272 6657
Fax: +1 714 821 4641
<http://computer.org/cspress>
csbooks@computer.org

IEEE Service Center
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
Tel: +1 732 981 0060
Fax: +1 732 981 9667
<http://shop.ieee.org/store/>
customer-service@ieee.org

IEEE Computer Society
Asia/Pacific Office
Watanabe Bldg., 1-4-2
Minami-Aoyama
Minato-ku, Tokyo 107-0062
JAPAN
Tel: +81 3 3408 3118
Fax: +81 3 3408 3553
tokyo.ofc@computer.org

Individual paper REPRINTS may be ordered at: reprints@computer.org

Editorial production by Randall Bilof

Cover art production by Joe Daigle/Studio Productions

Printed in the United States of America by The Printing House



IEEE Computer Society
Conference Publishing Services
<http://www.computer.org/proceedings/>

LzLabs GmbH. Ex. 1017-144

Contents

Proceedings of the 29th Annual International Computer Software and Applications Conference

COMPSAC 2005

Message from the Standing Committee Chair	xiii
Message from the General Chair	xiv
Message from the Program Co-Chairs	xv
Committees	xvii
Reviewers	xix

Keynote Address I

Industry Trends and Software Assurance.....	3
<i>G. Spittle</i>	

Software Testing (I)

Incremental Class Testing from a Class Test Order.....	7
<i>Y. Labiche</i>	
Configuration-Oriented Symbolic Test Sequence Construction Method for EFSM	13
<i>S. Li, J. Wang, X. Wang, and Z.-C. Qi</i>	
Iterative Metamorphic Testing.....	19
<i>P. Wu</i>	
Software Test Selection Patterns and Elusive Bugs.....	25
<i>W. E. Howden</i>	

Process Management

DIG: A Tool for Software Process Data Extraction and Grooming.....	35
<i>S. D. Miller, A. P. Mathur, and R. A. DeCarlo</i>	
A Defect Estimation Approach for Sequential Inspection Using a Modified Capture-Recapture Model.....	41
<i>C.-P. Chang, J.-L. Lv, and C.-P. Chu</i>	
Parallel Changes: Detecting Semantic Interferences.....	47
<i>G. L. Thione and D. E. Perry</i>	

Software Architecture and Metric

A Productivity Metric Based on Statistical Pattern Recognition	59
<i>J. L. Sharpe and J. W. Cangussu</i>	
Optimizing and Simplifying Software Metric Models Constructed Using Maximum Likelihood Methods	65
<i>V. K. Y. Chan and W. E. Wong</i>	
Structure Analysis for Dynamic Software Architecture Based on Spatial Logic.....	71
<i>T. Han, T. Chen, and J. Lu</i>	
Performability Analysis of a Pipeline Software Architecture	77
<i>S. S. Gokhale and S. Yacoub</i>	

Panel

High-Assurance Software for Mobile Tele-Health Systems

Moderator: *J. C. Lacal*
Panelists: *F. Del Poxo*
R. S. H. Istepanian
D. J. McClure

High-Assurance Software Design for Mobile Tele-Health Systems.....	85
<i>J. C. Lacal</i>	
Wireless Security for Personalized and Mobile Healthcare Services.....	86
<i>R. S. H. Istepanian</i>	

Software Reengineering and Service-Based Systems

An Extended Approach to Improving the Semantic Interoperation Among Reuse Repositories.....	89
<i>Y. Pan, L. Wang, L. Zhang, B. Xie, and F. Yang</i>	
An Empirical Study of a Reverse Engineering Method for Aggregation Relationship Based on Operation Propagation	95
<i>D. Yeh, P.-C. Sun, W. Chu, C.-L. Lin, and H. Yang</i>	
Adaptive Testing, Oracle Generation, and Test Case Ranking for Web Services.....	101
<i>W.-T. Tsai, Y. Chen, R. Paul, H. Huang, X. Zhou, and X. Wei</i>	
Situation-Awareness for Adaptive Coordination in Service-Based Systems	107
<i>S. S. Yau, D. Huang, H. Gong, and H. Davulcu</i>	

Middleware and Agent-Based Systems

Self-Configuring Communication Middleware Model for Multiple Network Interfaces	115
<i>N. Mohamed</i>	
Using Aspect Oriented Techniques to Support Separation of Concerns in Model Driven Development.....	121
<i>A. Solberg, D. Simmonds, R. Reddy, S. Ghosh, and R. France</i>	
Agent Oriented Programming Based on SLABS	127
<i>J. Wang, R. Shen, and H. Zhu</i>	

Intention Scheduling for BDI Agent Systems	133
<i>S.-B. Yan, Z.-N. Lin, H.-J. Hsu, and F.-J. Wang</i>	

Security and Privacy (I)

Detecting Intrusions Specified in a Software Specification Language	143
<i>M. F. Raihan and M. Zulkernine</i>	
SecureC: Control-Flow Protection Against General Buffer Overflow Attack	149
<i>H. Nishiyama</i>	
Buffer Overrun Prevention Through Component Composition Analysis	156
<i>D. Llewellyn-Jones, M. Merabti, Q. Shi, and B. Askwith</i>	
Efficiently Managing Security Concerns in Component Based System Design	164
<i>A. Masood, S. Sedigh-Ali, and A. Ghafoor</i>	

Panel

Industry and Academia: Can They Agree on Best Practice Within Software Engineering?

Moderator: *J. B. Thompson*

Panelists: *M. Boasson*

D. Simmons

S. Chen

P. Bourque

Industry and Academia: Can They Agree on Best Practice Within Software Engineering?	173
<i>J. B. Thompson</i>	
Best Practices in Software Engineering: Does Anybody in Industry or Academia Care?	174
<i>S. Chen</i>	
Principles Should Underlie Standards and Best Practices	175
<i>P. Bourque</i>	

Requirement and Design (I)

The Vital Few Versus the Trivial Many: Examining the Pareto Principle for Software	179
<i>M. Gittens, Y. Kim, and D. Godwin</i>	
Transitioning from Product Line Requirements to Product Line Architecture	186
<i>J. Savolainen, I. Oliver, M. Mannion, and H. Zuo</i>	
Designing Controllers for Reachability	196
<i>C. C. Seceleanu</i>	
Melody Classification Using EM Algorithm	204
<i>Y. Yoshihara and T. Miura</i>	

Formal Methods

A Formal Approach to Designing a Class-Subclass Structure Using a Partial-Order on the Functions	213
<i>S. Kundu and N. Gwee</i>	
A Family of Extended Fuzzy Description Logics	221
<i>Y. Li, B. Xu, J. Lu, D. Kang, and P. Wang</i>	
Extended Symbolic Transition Graphs with Assignment	227
<i>W. Deng and H. Lin</i>	
MetaWSL and Meta-Transformations in the FermaT Transformation System.....	233
<i>M. Ward and H. Zedan</i>	

Networked, Distributed, and Embedded Systems (I)

Using AOP to Monitor and Administer Software for Grid Computing Environments	241
<i>M. Grechanik, D. E. Perry, and D. Batory</i>	
Nearest Neighbor Queries on Extensible Grid Files Using Dimensionality Reduction	249
<i>R. Miyoshi, T. Miura, and I. Shioya</i>	
Power Transmission Control Using Distributed Max Flow	256
<i>A. Armbruster, M. Gosnell, B. McMillin, and M. L. Crow</i>	

Plenary Panel

Security and Privacy in Distributed Collaborative Systems

Chair:	<i>S. S. Yau</i>
Panelists:	<i>P. Bonatti</i>
	<i>D. Feng</i>
	<i>J. Xiang</i>
	<i>B. Thuraisingham</i> (see the position paper on page 561)

Security and Privacy in Collaborative Distributed Systems.....	267
<i>S. S. Yau</i>	
Rule Languages for Security and Privacy in Cooperative Systems	268
<i>P. Bonatti</i>	
Experiences on Intrusion Tolerance Distributed Systems.....	270
<i>D. Feng and J. Xiang</i>	

Internet Web Systems

Tree-Map Visualization for Web Accessibility.....	275
<i>J. Bailey and E. Burd</i>	
On Personal Data License Design and Negotiation	281
<i>Y.-J. Joung, C. Yen, C.-T. Huang, and Y.-J. Huang</i>	
Adapting Single-Request/Multiple-Response Messaging to Web Services.....	287
<i>M. Ruth, F. Lin, and S. Tu</i>	

A Web Data Extraction Description Language and Its Implementation	293
<i>I.-C. Wu, J.-Y. Su, and L.-B. Chen</i>	

Verification and Validation

A Blocking-Based Approach to Protocol Validation	301
<i>Y. Lei, D. Kung, and Q. Ye</i>	
An Empirical Performance Study for Validating a Performance Analysis Approach: PSIM	307
<i>J. Xia, Y. Ge, and C. K. Chang</i>	
Empirical Validation of Website Timeliness Measures	313
<i>Y. Zhang, H. Zhu, and S. Greenwood</i>	
Hybrid Intelligent Systems: Selecting Attributes for Soft-Computing Analysis.....	319
<i>P. Pattaraintakorn, N. Cercone, and K. Naruedomkul</i>	

Panel

Model-Based Software Testing and Verification

Moderator:

Panelists:

A. P. Mathur

M. P. E. Heimdahl

A. Jain

T. H. Tse

F. Belli

Model-Based Software Testing and Verification	329
<i>A. P. Mathur</i>	
Model-Based Testing: Challenges Ahead	330
<i>M. P. E. Heimdahl</i>	
Model-Based Testing for Enterprise Software Solutions	331
<i>A. Jain</i>	
Research Directions on Model-Based Metamorphic Testing and Verification.....	332
<i>T. H. Tse</i>	

Mobile and Pervasive/Ubiquitous Computing

A Probabilistic Approach to the Anxious Home for Activity Monitoring	335
<i>G. West, S. Greenhill, and S. Venkatesh</i>	
Exception Handling in Coordination-Based Mobile Environments.....	341
<i>A. Iliasov and A. Romanovsky</i>	
A User-Adaptable User Interface Model to Support Ubiquitous User Access to EIS Style Applications	351
<i>J. Davis, A. Tierney, and E. Chang</i>	
A Dynamic Context-Conflict Management Scheme for Group-Aware Ubiquitous Computing Environments	359
<i>I. Park, D. Lee, and S. J. Hyun</i>	

21st-Century Global Software Development and EducationModerator: *D. B. Simmons*Panelists: *C. Nelson**V. Livschitz**J. H. Cross II**X. Liu**J. B. Thompson*

Twenty-First Century Software Production Paradigms	367
<i>D. B. Simmons</i>	
SSI, Academia and Industry Collaboration.....	368
<i>C. Nelson</i>	
Conquering the Crisis in Software Engineering.....	369
<i>V. Livschitz</i>	
21st-Century Global Software Development and Education Position Statement.....	370
<i>J. H. Cross II</i>	
Collaborative Global Software Development and Education	371
<i>X. Liu</i>	
Future Software Development and Education—A U.K. View	372
<i>J. B. Thompson</i>	

Requirement and Design (II)

A Tool and Case Study for Specification-Based Program Review	375
<i>F. Nagoya, S. Liu, and Y. Chen</i>	
Tridirectional Computed Chaining: An Efficient Hashing Algorithm for Limited Space Applications	381
<i>M. Loeb and A. L. Tharp</i>	
A Requirement Engineering Approach for Designing XML-View Driven, XML Document Warehouses	388
<i>V. Nassis, R. Rajugan, T. S. Dillon, and W. Rahayu</i>	
Structured Object-Oriented Co-Analysis/Co-Design of Hardware/Software for the FACTS Power System	396
<i>M. Ryan, S. Markose, X. Liu, B. McMillin, and Y. Cheng</i>	

Reliable Systems

Gompertz Software Reliability Model and Its Application.....	405
<i>K. Ohishi, H. Okamura, and T. Dohi</i>	
Software Reliability Growth Model Considering Testing Profile and Operation Profile	411
<i>J. Zhao, H.-W. Liu, G. Cui, and X.-Z. Yang</i>	
RIES—Internet Voting in Action.....	417
<i>E. Hubbers, B. Jacobs, and W. Pieters</i>	

Two Variations to the mCESG Pollsterless E-Voting Scheme	425
<i>T. Storer and I. Duncan</i>	

Keynote Address II

Towards a Model of Seamless Mobility—User Value and Security in Computing, Content, and Communications.....	433
<i>J. Srinivas and S. Chen</i>	

Software Testing (II)

Towards Describing Black-Box Testing Methods as Atomic Rules.....	437
<i>T. Murnane, R. Hall, and K. Reed</i>	
Testing the Semantics of W3C XML Schema	443
<i>J. B. Li and J. Miller</i>	
Goal-Oriented Test Data Generation for Programs with Pointer Variables.....	449
<i>A. Gotlieb, T. Denmat, and B. Botella</i>	
Coverage-Directed Test Generation with Model Checkers: Challenges and Opportunities	455
<i>G. Devaraj, M. P. E. Heimdahl, and D. Liang</i>	
Testing Coverage Analysis for Software Component Validation	463
<i>J. Gao, R. Espinoza, and J. He</i>	

Dependability and Fault Tolerance

Availability Evaluation of Hardware/Software Systems with Several Recovery Procedures.....	473
<i>S. A. Vilkomir, D. L. Parnas, V. B. Mendiratta, and E. Murphy</i>	
A Comparison of Network Level Fault Injection with Code Insertion	479
<i>N. Looker, M. Munro, and J. Xu</i>	
The Coordinated Recovery of Data Service and Transaction Service in J2EE.....	485
<i>T. Liu, G. Huang, G. Fan, and H. Mei</i>	
A Low-Latency Checkpointing Scheme for Mobile Computing Systems.....	491
<i>G. Li and L. Shu</i>	

Panel

How Democratic Will E-Democracy Be?

Moderators:	<i>S. Anderson and M. Felici</i>
Panelists:	<i>I. Brown</i>
	<i>J. Bus</i>
	<i>A. Modi</i>
	<i>P. Ryan</i>

How Democratic Will E-Democracy Be?.....	499
<i>S. Anderson and M. Felici</i>	
Who Is Enfranchised by Remote Voting?.....	500
<i>I. Brown</i>	

Prospects for E-Voting	501
<i>P. Ryan</i>	
How Democratic Will E-Democracy Be?	502
<i>J. Bus</i>	

Security and Privacy (II)

Automated Negotiation of Privacy Contracts	505
<i>M. Maaser and P. Langendoerfer</i>	
A Novel Method for Protecting Sensitive Knowledge in Association Rules Mining	511
<i>E. T. Wang, G. Lee, and Y. T. Lin</i>	
Protecting Users Against Phishing Attacks with AntiPhish	517
<i>E. Kirda and C. Kruegel</i>	
A Performance-Based Grid Intrusion Detection System	525
<i>F.-Y. Leu, J.-C. Lin, M.-C. Li, and C.-T. Yang</i>	

Networked, Distributed, and Embedded Systems (II)

Scoped Broadcast in Dynamic Peer-to-Peer Networks	533
<i>H.-C. Hsiao and C.-T. King</i>	
Character Recognition System for Cellular Phone with Camera	539
<i>K. S. Bae, K. K. Kim, Y. G. Chung, and W. P. Yu</i>	
Towards a Contract Aware CORBA Component Container	545
<i>E. Teiniker, R. Lechner, G. Schmoelzer, C. Kreiner, Z. Kovács, and R. Weiss</i>	
New Method for Dispatching Waiting Logical Processors in Virtual Machine Systems	551
<i>H. Umeno, M. Kiyama, T. Fukunaga, and T. Kubo</i>	
Trust Management in a Distributed Environment (<i>This is the position paper of Bhavani</i> <i>Thuraisingham, a panelist of the plenary panel, which arrived late. Please see Plenary Panel:</i> <i>Security and Privacy in Distributed Collaborative Systems, page 267.</i>)	561
<i>B. Thuraisingham</i>	

Author Index	563
---------------------------	-----

New Method for Dispatching Waiting Logical Processors in Virtual Machine System

Hidehori Umeno,
Dep. of CS, Fac. of Eng.,
Kumamoto University
umeno@cs.kumamoto-u.ac.jp

Masato Kiyama,
Dep. of CS, Fac. of Eng.,
Kumamoto University
masato@cs.kumamoto-u.ac.jp

Takafumi Fukunaga
Kumamoto Pref. College of
Technology
fukunaga@kumamoto-pct.ac.jp

Takashige Kubo,
Hitachi System & Service,
Ltd.
taka11k@dolphin.ocn.ne.jp

Abstract

A virtual machine system can run multiple conventional operating systems (OSs) in a single real host computer. A virtual machine is a logical computer with almost the same architecture as the host, and may contain several logical processors. A hypervisor is a control program to control this virtual machine system. Traditionally, the hypervisor has to receive an I/O interrupt pending for a waiting logical processor, and to simulate the I/O interrupt, consequently incurring the simulation overhead of the I/O interrupt. To avoid this overhead we present a new method which introduces a new self-wait state different from the conventional wait state, presents a new instruction for the hypervisor to detect the I/O interrupts pending for the logical processors in the self-wait state, and dispatches those logical processors on the detection ahead of the ready queue. This new method has eliminated the simulation overhead of those I/O interrupts, and enhanced the system performance to the near native.

1. Introduction

A virtual machine system can concurrently run multiple operating systems (OSs) in a single real host computer system [1], [2], [3], [4]. The virtual machine system has been used in mainframes, nowadays even in personal computers, as VMware [5], Virtual PC, etc. A Virtual Machine (VM), which may be called a guest, is a logical computer which has almost the same architecture as the real host computer. The real host computer here may be a uniprocessor (UP) system or a symmetric multiprocessor (SMP) system. As the SMP has multiple real processors, each of which has the same functions and shares its main memory, so the VM may have multiple logical processors in the same way,

which is called a guest multiprocessor system.

Recently, many virtual machine systems are supported in the personal computers with x86 architecture. Some of them give users only the partial hardware architecture of the real host computer, providing users with the simplified logical devices [7], [8]. On the other hand, some of them give users the full architecture by simulating all the guest I/Os [5]. As a result they have no capability of running the existing legacy operating systems intact, or they have the high I/O simulation overhead.

We address the VMs that have almost the same architecture as the real host computer and that are capable of running multiple existing OSs concurrently and with high performance due to the hardware assists for virtual machines [2], [3], [4]. They will be run as production systems.

This paper presents the virtual machine system's performance improvement, its development, and its evaluation. A hypervisor is the program which controls the whole virtual machine system, allocates real resources (real processors, real memory, and real I/O devices) to each VM, and schedules its logical processors.

The scheduler of the hypervisor schedules the logical processors in the similar way as a traditional scheduler of a usual OS schedules its processes.

That is, it gives each logical processor one of the three states, which are running state, wait state, and ready state [9].

Generally, an interrupt to the hypervisor is called a host interrupt, which may be caused by the hypervisor itself such as a host timer interrupt, or may be caused by a VM and should be reported to the target VM. When a VM has the multiple logical processors, its I/O interrupt is reported to any one of them.

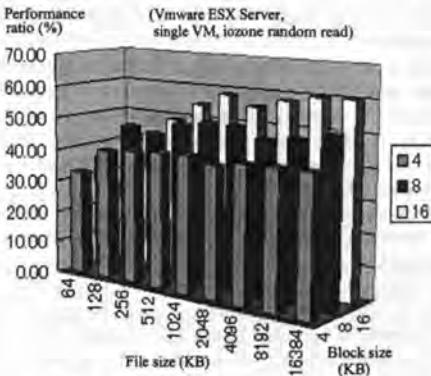
Conventionally, an I/O interrupt pending for a VM is directly executed by hardware when its target logical processor is running, therefore, it causes no

simulation overhead in this case [2], [4]. On the other hand, when all the target logical processors of that interrupt are in the wait state, it becomes a host interrupt. That is, the scheduler of the hypervisor recognizes the pending guest I/O interrupt through the host interrupt. Then, it simulates that interrupt to its target logical processor, and changes its state from the wait state to the ready state. This conventional scheduling is similar to the method a usual OS takes to schedule its processes [9], and causes the high simulation overhead of that guest I/O interrupt.

The hypervisor runs in the native mode, while a VM runs in the vm mode, which is not only a software mode but also a hardware supported mode [2], [3]. Changing the mode from vm to native or vice versa takes more than 100 times a usual instruction execution time. This is because the mode change contains the setting of many hardware control registers, resulting in suspending the hardware parallel execution temporally.

Conventionally an I/O interrupt pending for a target VM may cause a host interrupt when it (i.e. the target VM) is not running, while other VMs may be running and then intercepted by that host interrupt. If many guest I/O interrupts occur as host interrupts, they cause many mode changes from vm to native and vice versa, causing high transition overhead.

The above-mentioned I/O simulation overhead will be very high when a guest OS (i.e. an OS in a VM) runs heavy I/O workloads [5]. As an example of I/O simulation overhead we measured the I/O performance of VMware ESX Server, showing it in Figure 1 which indicates the VM I/O performance falls to 40 % - 60 % of the native performance in such workloads (i.e. the iotzone random read). This overhead is due to both the guest I/O instruction simulation and the guest I/O interrupt simulation.



used for everyday production, where each VM has the high performance except for several testing VMs which may coexist in the system.

In mainframe computers the virtual machine system hypervisor runs in the native mode, while a VM runs in the vm mode supported by hardware [1], [2], [3]. When a logical processor has issued a system wait instruction, the overall system must not fall into the wait state due to it. Because of this, hardware detects such an instruction issued and suppresses its execution, and transfers control to the hypervisor which then puts the logical processor in the wait state and schedules the other logical processors.

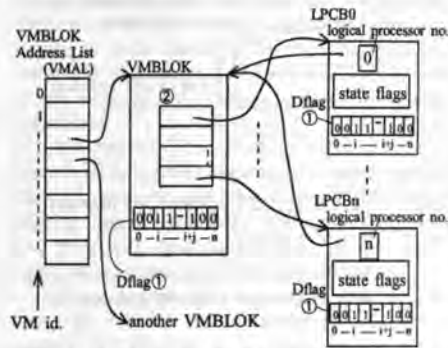


Fig. 3. Example of VM control blocks

On the other hand personal computers with IA32 architecture have no virtual machine hardware features such as the vm mode and the interception of the sensitive instructions. The hypervisor on those personal computers gives users the full architecture of the host by translating the guest binary codes containing sensitive instructions, tracking the guest segmented architecture by making shadow descriptor tables [7], which are modified copies of the guest descriptor tables. These simulations reduce the guest performance greatly for many processings such as the I/O processings shown in Figure 1.

Figure 3 shows an example of a configuration of various virtual machine control blocks, such as a VMBLOCK which stands for a control block for a VM, $LPCB_i$ ($i = 0, 1, \dots, n$) each of which stands for a logical processor thereof and contains state flags for managing its states, and a VMAL which stands for address list for VMBLOCKS. The hypervisor creates and manages these control blocks in main memory, scheduling logical processors using those LPCBs, managing their state flags, and defining Dflag (①) explained in section 3. This configuration is conventional and not new.

The relation between the hypervisor and the guest

OSs is similar to that between an OS and its application processes [9]. Contrary to this similarity, the scheduler of the hypervisor differs from a process scheduler of an OS in the following three points.

(1) An interrupt may be reported to a running logical processor directly, that is, by hardware and without intervention of the hypervisor.

(2) A logical processor in the running state will keep running immediately after it has accepted its interrupts directly.

(3) Conventionally if an interrupt request occurs and no logical processors of its target are running, it becomes a host interrupt, then, the hypervisor simulates it according to the hardware architecture and reports it to one of the target logical processors.

The differences (1) and (2) enhance the total system performance because of no intervention of the hypervisor, while the last difference (3) causes the guest I/O interrupt simulation and the mode change overhead stated in section 1, and reduces the system performance. We propose a new scheduling method to reduce those overheads in the following section.

3. New scheduling method

We propose a new scheduling method consisting of the following four methods:

- (1) adding a new self-wait state different from the conventional wait state,
- (2) detecting the I/O interrupts pending for the logical processors in the self-wait state by using a new instruction,
- (3) scheduling the logical processors in the self-wait state,
- (4) searching for workloads in an active wait state with two stages when workloads have been exhausted.

3.1. Self-wait state

If a running logical processor falls into the wait state, it is due to (1) the exhaustion of its workloads, or due to (2) the hypervisor commands to stop it.

The operators do not issue the hypervisor commands to the logical processor during its production running, while they may issue them during its test running, which means a guest OS is tested in the logical processor.

In the former case, an OS in the logical processor usually falls into the system wait state, which will be waked up by an interrupt to it. In the latter case, the logical processor should be started also by some hypervisor commands. To distinguish these two kinds of wait state, we call the former the self-wait state.

Figure 4 shows conventional states (①, ②, ③,

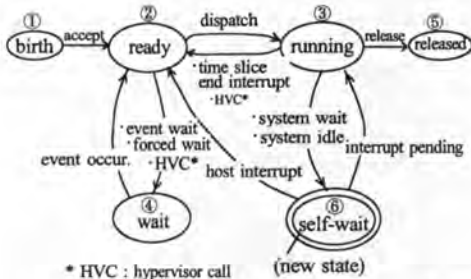


Fig. 4. Self-wait state of logical processors

(4), (5) and new state (6), that is, birth state (1), ready state (2), running state (3), released state (5), conventional wait state (4), and the new self-wait state (6). The logical processors in the self-wait state enter a self-wait queue, which is separated from a conventional wait queue, which consists of the logical processors in the conventional wait state (4). We present a method for detecting a VM's pending I/O interrupts by scanning this self-wait queue. This separation makes the scan quick. The interrupt delay is reduced because the scheduler of the hypervisor scans the self-wait queue ahead of the ready queue.

3.2. Conventional direct I/O execution method

Conventionally, I/O instructions issued by a guest OS are directly executed by hardware when they are issued to subchannels dedicated to the guest OS, and their interrupts are also directly executed during its running [2], [4]. This makes its performance close to that of the native machine. We premise that this direct execution method is already used in the real host machine. The conventional direct I/O execution method has been supported to the running logical processors. The I/O interrupts pending for VMs become host interrupts when all their target logical processors are in the wait state or in the ready state as stated in section 1. Our new method applies this direct execution method also to the self-wait logical processors though it may additionally delay the I/O interrupts to them.

Here, to clarify our method, we simply explain the conventional direct execution method having the following features,

(1) hardware logic recognizes the I/O interrupts pending for VMs without the intervention of the hypervisor,

(2) hardware logic judges whether or not a running logical processor is enabled for its pending I/O interrupts without the intervention of the hypervisor, (3) hardware logic informs a running logical processor of its I/O interrupts when it is enabled for them without the intervention of the hypervisor.

To support the feature (1), we allocate I/O interrupt priorities exclusively to VMs [2], [3]. Figure 5 shows I/O interrupt request queues. It illustrates an example of the exclusive allocation of the I/O

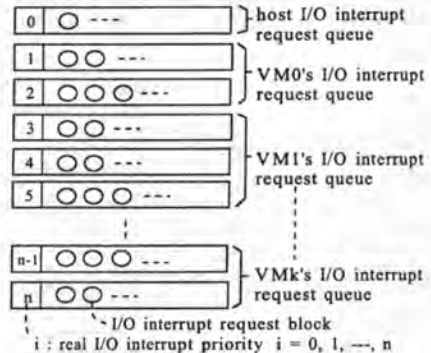


Fig. 5. I/O interrupt request queues (Conventional)

interrupt priorities 0, 1, 2, ..., n. It allocates priorities 1, 2 to VM0, priorities 3, 4, 5 to VM1, ..., priorities n-1, n to VMk, and allocates priority 0 to the hypervisor. An I/O interrupt request with priority (i) is enqueued into the request queue with priority (i). These I/O interrupt request queues are composed in real memory and managed by hardware, such as CPUs and I/O processors. With this queuing, hardware can recognize the I/O interrupt requests pending for VMs. For example, if an I/O interrupt request is enqueued into the queue with priority (i), and if the priority (i) is dedicated to VM1, the interrupt belongs to the VM1. As a result, hardware can recognize that the VM1 has the pending I/O interrupt requests.

This method does not require so many interrupt priorities, indeed, most OSs only use a few (probably two or three) priorities. This is because they will poll any I/O interrupts using some I/O instructions. These facts lead to this method which allocates a few interrupt priorities exclusively to VMs.

To support the feature (2) and (3), hardware has to have an interrupt control logic for a running VM which contains an I/O interrupt mask register and an I/O interrupt priority mask register. Several variations have been supported for those features, such as giving only one priority to each VM [3], or giving multiple priorities to each VM [2], or giving all the priorities

to each VM [4].

This conventional direct execution method can be applied only to the running logical processors. We apply it also to the self-wait logical processors.

3.3. Detecting I/O interrupts pending for self-wait logical processors

An I/O interrupt request occurs even when its target logical processor is in the wait state. Conventionally, it brings about a host interrupt. We present a method for suppressing this host interrupt. The I/O interrupt priorities are exclusively allocated to VMs as shown in Figure 5. We set the host masks of those exclusively allocated priorities always to be 0. This mask setting always suppresses the host I/O interrupts with those priorities. The pending I/O interrupts with those priorities are directly executed when one of its target logical processors are dispatched.

Conventionally, no scheduler dispatches the logical processors in the wait state or in the self-wait state. This is natural because the forced dispatching of the waiting logical processors will activate no activities and they will immediately fall into the wait state again, only increasing scheduling overhead.

After a logical processor falls into the self-wait state, its external system activities will generate a pending I/O interrupt or a pending timer interrupt or another interrupt pending for itself. When the logical processor is dispatched, those interrupts pending for itself can be directly executed by the hardware as long as the hardware can recognize those pending interrupts.

If the hypervisor is provided with a hardware function that can quickly detect the I/O interrupts pending for a VM, it can use the function to detect them. When detecting an interrupt pending for a self-wait logical processor, and it is enabled for that

interrupt, the hypervisor can dispatch it. Then, that interrupt request will be directly executed for it by the hardware.

Based on this consideration, we propose a new machine instruction called "Only Test Pending I/O Interrupts (OTPI)" shown in Figure 6, which detects an I/O interrupt pending with the priority i ($i = 0, 1, \dots, n$), which is an input parameter stored in a register R1, storing non-0 in a register R2 when detected, or 0 otherwise as a result. This instruction can be defined in any hardware architecture that has the I/O interrupt priorities $0, 1, \dots, n$ into which hardware enqueues the I/O interrupt requests.

Users allocate real I/O interrupt priorities exclusively to VMs using hypervisor commands. The hypervisor composes logical control blocks shown in Figure 3, which contains a Dflag (①), which indicates real interrupt priorities dedicated to the VM. By accessing it, the hypervisor can get those priorities, and can use this instruction with them, and can determine whether the VM has any pending I/O interrupts with those priorities.

The hardware usually has I/O interrupt request queues shown in Figure 5. This instruction is very simple and only checks the queue with the designated priority, and much faster than the I/O interrupt execution. This is because it only checks the designated queue and does not dequeue any I/O interrupt request element, while the I/O interrupt execution dequeues its request element and carries out the subsequent hardware interrupt procedures. Thus, the hypervisor can quickly determine whether a waiting logical processor has an I/O interrupt pending with its dedicated priorities without any host I/O interrupt for them.

We say that a logical processor is in the "self-wait-ready state," when it is in the self-wait state, at the same time, it has an I/O interrupt pending for itself, and it is enabled for that interrupt.

Conventionally, IBM provides an instruction called "Test Pending Zone Interrupt (TPZI)" [3], which checks the I/O interrupts of the designated guest, selectively can complete here, or continuously dequeues the I/O interrupt requests elements, and informs the hypervisor of the interrupt information. We think that this TPZI instruction is probably used to reap the guest I/O interrupt requests in the hypervisor of IBM's virtual machine system. This instruction seems to correspond to "Test Pending Interrupt (TPI)" instruction [12], [13] that native OSs use to check the pending I/O interrupts in the hardware queue and to dequeue (i.e. reap) the I/O interrupt request elements in the queue. Our OTPI instruction only tests the I/O interrupts pending in the queue and does not dequeue any request elements, therefore, much faster than those instructions. We do

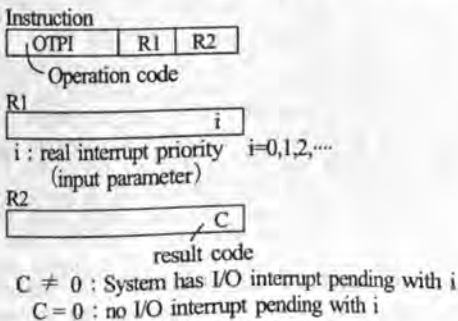


Fig. 6. Only Test Pending I/O Interrupt (OTPI) instruction

Some waiting logical processors may come into the ready queue by host interrupts which may be host timer interrupts or may be host I/O interrupts with shared priorities. When finding the ready queue counter non-zero, the hypervisor exits from this stage (wls-1) and goes to the scheduler, when finding the ready queue counter zero and the self-wait queue counter non-zero, it exits from this stage, and goes to the second stage (wls-2), and when finding both counters zero, it loops between them.

In the second stage (wls-2) the hypervisor scans the self-wait queue, and checks if each logical processor has I/O interrupts pending for itself. The hypervisor checks the self-wait logical processor (we call this A) by issuing an OTPI instruction whose operand is an I/O interrupt priority dedicated to the logical processor A, and sets its self-wait-ready flag if it has the I/O interrupt pending for itself and is enabled for that interrupt.

In that case the hypervisor sets the self-wait-ready flags of all the self-wait logical processors that are brothers of the logical processor A, that is, belong to the same VM. This is because there may be several I/O interrupts pending for the VM in the interrupt request queues with its dedicated priorities (Refer to Figure 5).

The hypervisor can easily find dedicated I/O priorities by seeing the Dflag (①) of the logical processor as shown in Figure 3, where priorities $i, i+1, \dots, i+j$ ($j \geq 0$) are dedicated to a VM (represented as VMBLOCK), and to logical processors $0, 1, \dots, n$ (as LPCB0, LPCB1, $\dots$, LPCBn, respectively). It can also easily find all the logical processors belonging to the same VM by seeing its pointers ② in Figure 3. Thus when finding at least one self-wait-ready logical processor, the hypervisor exits from the stage 2 workload search and goes to the scheduler to dispatch it, when finding no such logical processors, it returns to the first stage (wls-1).

The pending I/O interrupts may occur asynchronously to the real processors. The hypervisor has to get the logical processor's most current state. Therefore, it has to issue the OTPI instruction each time it checks if the logical processor has a pending I/O interrupt. It is not sufficient for the hypervisor to read all the pending mask bits only once of all self-wait logical processors because the values of their mask bits may vary momentarily. Moreover, the hypervisor should schedule them in the order of their self-wait states and their pending I/O interrupts.

4. Related art and architecture

Our proposed method schedules and dispatches the self-wait logical processors. A conventional OS's

scheduler does not dispatch the waiting processes, usually accepting interrupts for them, changing their states from the wait to the ready, after that, dispatching only the ready state processes, thus having no methods for scheduling and dispatching the waiting processes directly.

Active wait method is supported conventionally [10]. It opens all the interrupt masks, causing any kind of interrupts, resulting in the increase of their simulation overhead in the VM environment. Instead of this, our active wait method disables for the I/O interrupts pending for VMs with dedicated interrupt priorities, detecting them without host interrupts, responsive to the detection, scheduling and dispatching the VMs.

Conventionally, Denali [7] has the lazy interrupt processing and Xen [8] has the batch processing for interrupts. In both methods the hypervisor accepts I/O interrupts pending for VMs, translating them into simple events, and reporting them at the scheduling of the VM. This is the I/O simulation and will cause high simulation overhead.

Contrary to this, in our method the hypervisor never accepts host I/O interrupts pending for a VM with dedicated interrupt priorities. The hypervisor detects those I/O interrupts pending for a waiting logical processor of a VM by using the new OTPI instruction, which can quickly detect them. After detecting them and checking if it is enabled for them, the hypervisor schedules it, then, they are directly executed in the vm mode hardware, thus causing no I/O simulation overhead.

Our method also has the conventional scheduling delay, and moreover may have the additional interrupt delay which is at most one scheduling time quantum due to the masking of the host I/O interrupts as described in section 3.4. We premise that our virtual machine system has the small number of VMs, that is, having at most ten VMs in the system. Therefore, these scheduling delays cause no problem because of the small time slice such as 1ms -10ms. The number of VMs is small because the guest OSs have almost the full use of the hardware and software functionalities and can run in near native performance, and will have the scalability to the number of their own processes.

Our method premises that the hardware architecture has a feature which supports direct I/O execution for VMs in their *running* states. It is widely supported in mainframe architectures [2], [4].

5. Implementation and performance evaluation

5.1. Implementation

As for hardware we have implemented the OTPI instruction, which detects I/O interrupts pending of the designated interrupt priority (See Figure 6). We used a hardware microcode program and two general purpose registers. This was very easy because the basic I/O interrupt queues were supported natively in hardware, only requiring additional control storage for its microcode.

As for software we have implemented the self-wait state, the mask setting to disable for the host I/O interrupts pending for VMs, the method for scheduling the self-wait logical processors, and the workload searching with two stages.

We have evaluated our scheduling method consisting of these proposed methods by measuring the performance for several workloads. Table 1 summarizes the performance comparison between the conventional host interrupt method and our proposed method. We describe the measurement method and the performance data in the following sections.

5.2. Measurement method

We address the users who daily operate multiple VMs concurrently under a single real host SMP. Several users have very large size systems consisting of several SMPs each of which has many (up to 128) processors sharing very large size (up to 2048GB) main memory and concurrently runs several (up to tens of) VMs. On the other hand, many users have medium size systems each of which will be a 2 to 4-way SMP system and runs 2 to 4 VMs concurrently.

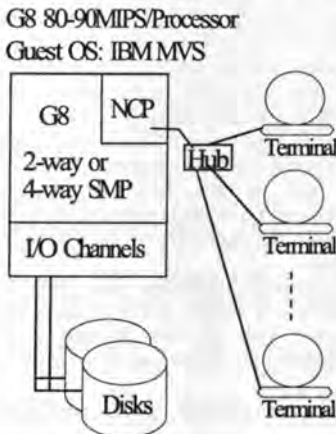


Fig. 8. Measurement Environment

Because our VMs can use almost the full functions of hardware and software, the number of VMs concurrently running is not so large, that is, up to ten. This is very different from the Denali VMs [7] and from the Xen VMs [8].

Considering that our virtual machine system has the small number of VMs running concurrently, we need not evaluate the scalability to the great number of VMs concurrently running. The scalability should be evaluated to the number of processes in each of the guest OSs. Here, the most important is the performance of each VM to the native machine.

We use the measurement environment shown in Figure 8, using a 4-way SMP, which consists of 4 processors sharing main memory, and a 2-way SMP, which consists of 2 processors sharing main memory.

Each processor has a 6.2ns machine cycle. This SMP has almost the same architecture as IBM System/390 [12]. The guest OS is the MVS, which is the OS of the IBM mainframe.

The number of logical processors of a VM is the same as that of the real host SMP. This is because the VM should have as many concurrently running logical processors as possible to improve its performance.

That is, when a real host is a 4-way SMP, we run 2 to 4 VMs, each of which is 4-way, that is, has 4 logical processors. When a 2-way SMP host, we run 2 VMs, each of which is 2-way, that is, has 2 logical processors.

The measurement of one case takes 1 to 3 hours including preparations, and measured workloads are I/O heavy jobs which seem to have the size and attributes comparable with real workloads. We use a performance monitor built in the guest OSs, summarizing the results in Table 1, explaining each case as follows.

Case 1 (#1 in Table 1) runs a 4-way SMP as a real host, in which running 2 VMs concurrently, each of which is 4-way, that is, has 4 logical processors. Its workloads are the Time Sharing System (TSS) jobs with 100 active terminals, and each VM has 50 terminals. We use a terminal simulator to generate the transactions each of which consists of a series of edit and its subcommands. We run the total system for about 15 minutes, getting performance data for two minutes (i.e. measurement interval in Table 1) of steady state during that running interval.

Table 1. Performance comparison of conventional method (a) with proposed method (b)

#	Workload	Item	Conventional method (a)	Proposed method (b)	Native mode (c)	b-a	(b-a)/a (%)	a/c (%)	b/c (%)	Comments
1	TSS 100 terminals	ETR ^{*1} (Ended/sec)	51.60	53.57	53.66	1.97	3.82	96.16	99.83	Host:4way
		Simul. ovh. ^{*3} (%)	3.21	1.85	—	-1.36	-42.37			2 VMs
		Sched. ovh. ^{*4} (%)	15.38	1.77	—	-13.61	-88.49			each VM:4way
		Interval ^{*5} (sec)	120.00	120.00	120.00	0.00				$\rho^* = 26\%$
2	I/O heavy	ETR ^{*2} (EXCPs/sec)	12.65	16.34	16.90	3.69	29.15	74.87	96.69	Host:2way
		Simul. ovh. ^{*3} (%)	12.95	7.43	—	-5.52	-42.63			2 VMs
		Sched. ovh. ^{*4} (%)	20.17	5.79	—	-14.38	-71.29			each VM:2way
		Interval ^{*5} (sec)	300.00	300.00	300.00	0.00				$\rho^* = 49\%$ (estimated from VM mode)
3	I/O heavy	ETR ^{*2} (EXCPs/sec)	13.30	16.31	16.70	3.01	22.63	79.64	97.66	Host:4way
		Simul. ovh. ^{*3} (%)	13.86	8.95	—	-4.91	-35.43			2 VMs
		Sched. ovh. ^{*4} (%)	42.96	8.47	—	-34.49	-80.28			each VM:4way
		Interval ^{*5} (sec)	300.00	300.00	300.00	0.00				$\rho^* = 25\%$

*: ρ : Native CPU utilization

Host Machine Cycle : 6.2ns

*1: ETR: External Throughput Ratio: Transactions ended / Real time one sec.

*2: ETR: External Throughput Ratio: Execute Channel Programs issued / Real time one sec.

*3: CPU utilization for simulating system wait inst. (Load PSW wait) and host timer interrupts that should be reported to guests

*4: CPU utilization for scheduling logical processors

*5: Measurement interval

Case 2 (#2 in Table 1) runs a 2-way SMP as a real host, in which running 2 VMs concurrently, each of which is 2-way, that is, has 2 logical processors. Its workloads are I/O heavy jobs issuing many Execute Channel Program (EXCP) macro-instructions, which are I/O macro-instructions, very frequently. We get the performance data for 5 minutes (i.e. measurement interval in Table 1) of steady state during that running interval.

Case 3 (#3 in Table 1) runs a 4-way SMP as a real host, in which running 2 VMs concurrently, each of which is 4-way, that is, has 4 logical processors. The other attributes are the same as the case 2.

5.3. Performance data

(1) Simulation Overhead (CPU utilization)

The simulation overhead belongs to each VM. It reduces the total system performance because it decreases of the CPU times for other VMs. Table 1 shows that the simulation overhead remarkably decreases compared with that of the conventional

method, that is, in CPU utilization, 3.21% to 1.85% in case 1, 12.95% to 7.43% in case 2, and 13.86% to 8.95% in case 3. This data means that our proposed method reduces the number of host I/O interrupts pending for the guest OSs, and the self-wait logical processors directly execute their own I/O interrupts at the time they have been dispatched.

(2) Scheduling Overhead (CPU utilization)

The scheduling overhead is the CPU utilization for scheduling logical processors, and if it is high, it reduces the total system performance. The overhead is remarkably reduced compared with that of the conventional method as Table 1 shows that 15.38% to 1.77% in case 1, 20.17% to 5.79% in case 2, and 42.96% to 8.47%. The high scheduling overhead of the conventional method is due to the high frequency of the host I/O interrupts pending for the guest OSs and its poor workload searching.

(3) External Throughput Ratio (ETR)

The external throughput ratio (ETR) is the number of transactions processed in case 1, or the number of EXCP (Execute Channel Program) macro-instructions issued in cases 2 and 3 in a real time second. This is

the performance from the point of user view.

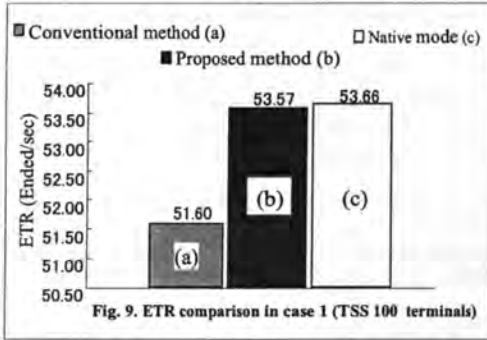


Fig. 9. ETR comparison in case 1 (TSS 100 terminals)

As for case 1 (TSS 100 terminals) Figure 9 shows a visible ETR improvement from 51.60 to 53.57.

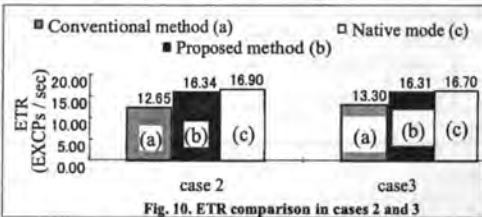


Fig. 10. ETR comparison in cases 2 and 3

As for cases 2 and 3 (I/O heavy jobs) Figure 10 shows a clear ETR improvement from 12.65 to 16.34 in case 2, and from 13.30 to 16.31 in case 3, additionally showing the ETR improvement reaching the near native performance.

These ETR improvements are almost due to the decrease of the I/O interrupt simulation for VMs and that of scheduling overhead.

6. Conclusions

We have presented a new method for scheduling logical processors in the self-wait state ahead of those in the ready state, and for searching workloads using two stages which scan the self-wait logical processors.

Having implemented this new method, we have evaluated it by measuring the performance for some workloads, and have confirmed their improvements by the performance data showing the decrease of the I/O interrupt simulation overhead and that of the

scheduling overhead. Moreover, we have confirmed the ETR performance improvement reaching the near native even for the I/O heavy workloads.

7. References

- [1] IBM, "Logical Partitioning", <http://www-1.ibm.com/servers/eserver/series/lpar/>.
- [2] Umeno, H., et al., "I/O execution method for a virtual machine system and system thereof", Hitachi, US 5109489, Priority Jan. 18, 1984.
- [3] Bean, George H., et al., "Logical Resource Partitioning of a Data Processing System", IBM, USP4843541, July 29, 1987 Filed.
- [4] Chou, Norman Cho-Chun, et al., "CPU expansive gradation of I/O interruption subclass recognition", US 752149, IBM, Filed Aug. 29, 1991.
- [5] Sugerman, J., et al., "Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor", VMware, Inc., Proc. of the 2001 USENIX Annual Tech. Conf., Boston, Ma., USA, June 25-30, 2001.
- [6] Scott W. Divine, et al., "Virtualization System Including a Virtual Machine Monitor for a Computer with a Segmented Architecture", US6397242B1, VMWare Inc., Filed Oct. 26, 1998.
- [7] Whitaker, A., Shaw, M., Gribble, Steven D., "Scale and Performance in the Denali Isolation Kernel", In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI 2002)*, ACM Operating Systems Review, Winter 2002 Special Issue, Boston, MA, USA, Dec. 2002, pp. 195-210.
- [8] Barham, P., Dragovic, B., Fraser, K., et al., "Xen and the Art of Virtualization", SOSP'03, October 19-22, 2003, Bolton Landing, NY, USA, pp. 164-177.
- [9] Stallings, W., "Operating Systems: Internal and Design Principles Fourth Edition", Prentice Hall (2000).
- [10] Blandy, G. O., et al., "Active wait", US4631674, IBM, Filed Feb. 5, 1985.
- [11] Umeno, H., et al., "Development of Methods for Reducing the Spins of Guest Multiprocessors", Trans. IPS Japan, Vol. 36, No. 3, pp. 681-696 (March, 1995).
- [12] IBM, "ESA/390 principles of operation", SA22-7201-08.
- [13] IBM, "z/Architecture Principles of Operation", SA22-7832.

New Method for Dispatching Waiting Logical Processors in Virtual Machine System

Hidenori Umeno,
Dep. of CS, Fac. of Eng,
Kumamoto University
umeno@cs.kumamoto-u.ac.jp

Masato Kiyama,
Dep. of CS, Fac. of Eng,
Kumamoto University
masato@cs.kumamoto-u.ac.jp

Takafumi Fukunaga
Kumamoto Pref. College of
Technology
fukunaga@kumamoto-pct.ac.jp

Takashige Kubo,
Hitachi System & Service,
Ltd.
takal1k@dolphin.ocn.ne.jp

Abstract

A virtual machine system can run multiple conventional operating systems (OSs) in a single real host computer. A virtual machine is a logical computer with almost the same architecture as the host, and may contain several logical processors. A hypervisor is a control program to control this virtual machine system. Traditionally, the hypervisor has to receive an I/O interrupt pending for a waiting logical processor, and to simulate the I/O interrupt, consequently incurring the simulation overhead of the I/O interrupt. To avoid this overhead we present a new method which introduces a new self-wait state different from the conventional wait state, presents a new instruction for the hypervisor to detect the I/O interrupts pending for the logical processors in the self-wait state, and dispatches those logical processors on the detection ahead of the ready queue. This new method has eliminated the simulation overhead of those I/O interrupts, and enhanced the system performance to the near native.

1. Introduction

A virtual machine system can concurrently run multiple operating systems (OSs) in a single real host computer system [1], [2], [3], [4]. The virtual machine system has been used in mainframes, nowadays even in personal computers, as VMware [5], Virtual PC, etc. A Virtual Machine (VM), which may be called a guest, is a logical computer which has almost the same architecture as the real host computer. The real host computer here may be a uniprocessor (UP) system or a symmetric multiprocessor (SMP) system. As the SMP has multiple real processors, each of which has the same functions and shares its main memory, so the VM may have multiple logical processors in the same way,

which is called a guest multiprocessor system.

Recently, many virtual machine systems are supported in the personal computers with x86 architecture. Some of them give users only the partial hardware architecture of the real host computer, providing users with the simplified logical devices [7], [8]. On the other hand, some of them give users the full architecture by simulating all the guest I/Os [5]. As a result they have no capability of running the existing legacy operating systems intact, or they have the high I/O simulation overhead.

We address the VMs that have almost the same architecture as the real host computer and that are capable of running multiple existing OSs concurrently and with high performance due to the hardware assists for virtual machines [2], [3], [4]. They will be run as production systems.

This paper presents the virtual machine system's performance improvement, its development, and its evaluation. A hypervisor is the program which controls the whole virtual machine system, allocates real resources (real processors, real memory, and real I/O devices) to each VM, and schedules its logical processors.

The scheduler of the hypervisor schedules the logical processors in the similar way as a traditional scheduler of a usual OS schedules its processes.

That is, it gives each logical processor one of the three states, which are running state, wait state, and ready state [9].

Generally, an interrupt to the hypervisor is called a host interrupt, which may be caused by the hypervisor itself such as a host timer interrupt, or may be caused by a VM and should be reported to the target VM.

When a VM has the multiple logical processors, its I/O interrupt is reported to any one of them.

Conventionally, an I/O interrupt pending for a VM is directly executed by hardware when its target logical processor is running, therefore, it causes no

simulation overhead in this case [2], [4]. On the other hand, when all the target logical processors of that interrupt are in the wait state, it becomes a host interrupt. That is, the scheduler of the hypervisor recognizes the pending guest I/O interrupt through the host interrupt. Then, it simulates that interrupt to its target logical processor, and changes its state from the wait state to the ready state. This conventional scheduling is similar to the method a usual OS takes to schedule its processes [9], and causes the high simulation overhead of that guest I/O interrupt.

The hypervisor runs in the native mode, while a VM runs in the vm mode, which is not only a software mode but also a hardware supported mode [2], [3]. Changing the mode from vm to native or vice versa takes more than 100 times a usual instruction execution time. This is because the mode change contains the setting of many hardware control registers, resulting in suspending the hardware parallel execution temporally.

Conventionally an I/O interrupt pending for a target VM may cause a host interrupt when it (i.e. the target VM) is not running, while other VMs may be running and then intercepted by that host interrupt. If many guest I/O interrupts occur as host interrupts, they cause many mode changes from vm to native and vice versa, causing high transition overhead.

The above-mentioned I/O simulation overhead will be very high when a guest OS (i.e. an OS in a VM) runs heavy I/O workloads [5]. As an example of I/O simulation overhead we measured the I/O performance of VMware ESX Server, showing it in Figure 1 which indicates the VM I/O performance falls to 40 % - 60 % of the native performance in such workloads (i.e. the iotzone random read). This overhead is due to both the guest I/O instruction simulation and the guest I/O interrupt simulation.

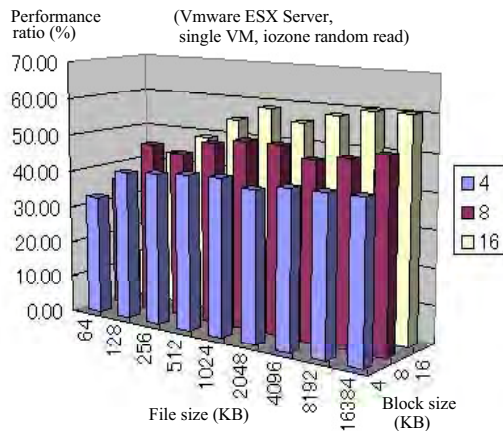


Fig. 1. VM I/O performance to native

To reduce those overheads, which consist of the

guest I/O interrupt simulation and the mode change overhead, we present a new method which introduces a new self-wait state different from the conventional wait state, presents a new instruction for the hypervisor to detect the I/O interrupts pending for the logical processors in the self-wait state, dispatches those logical processors in response to the detection ahead of the regular ready queue, and scans those logical processors to find out workloads by using the new instruction when workloads have been exhausted.

The detection and the finding workloads are carried out without any host I/O interrupt pending for VMs. This new method reduces those overheads and improves the performance of the virtual machine system to the near native even for heavy I/O workloads.

The system administrators will desire to run several VMs. The high I/O simulation overhead increases CPU utilization of the VM with heavy I/O workloads. This overhead reduces the CPU utilization for other VMs. Our method reduces I/O simulation overhead and gives benefit to the other VMs.

We state the concept of the virtual machine system in section 2, propose a new scheduling method in section 3, state the conventional related art and architecture in section 4, provide the experimental performance evaluation in section 5, and conclusions in section 6.

2. Virtual machine system

Figure 2 shows the configuration of the virtual machine system we are now considering, where virtual machines VM₀, VM₁, ..., VM_n are defined, each of which (i.e. VM_i) has logical processors LPi₀, LPi₁, ..., which share their main memory area. A virtual machine may have any number of logical processors. The number of logical processors may vary in each VM.

We consider that the virtual machine systems are

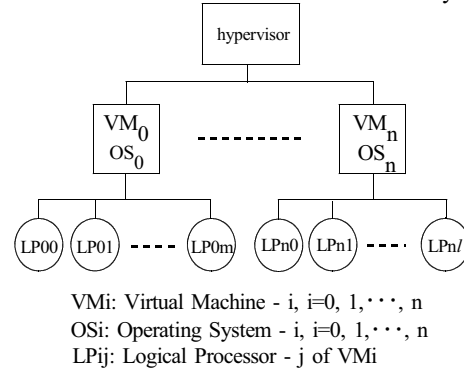


Fig. 2. Concept of virtual machine system

used for everyday production, where each VM has the high performance except for several testing VMs which may coexist in the system.

In mainframe computers the virtual machine system hypervisor runs in the native mode, while a VM runs in the vm mode supported by hardware [1], [2], [3]. When a logical processor has issued a system wait instruction, the overall system must not fall into the wait state due to it. Because of this, hardware detects such an instruction issued and suppresses its execution, and transfers control to the hypervisor which then puts the logical processor in the wait state and schedules the other logical processors.

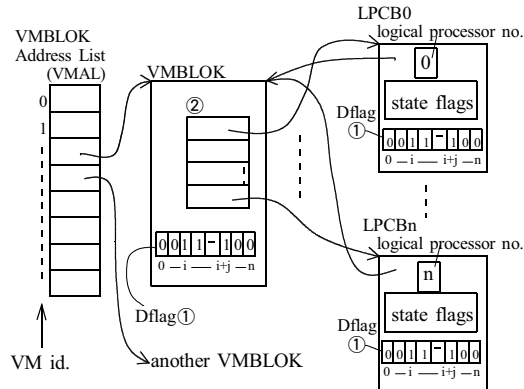


Fig. 3. Example of VM control blocks

On the other hand personal computers with IA32 architecture have no virtual machine hardware features such as the vm mode and the interception of the sensitive instructions. The hypervisor on those personal computers gives users the full architecture of the host by translating the guest binary codes containing sensitive instructions, tracking the guest segmented architecture by making shadow descriptor tables [7], which are modified copies of the guest descriptor tables. These simulations reduce the guest performance greatly for many processings such as the I/O processings shown in Figure 1.

Figure 3 shows an example of a configuration of various virtual machine control blocks, such as a VMBLOCK which stands for a control block for a VM, LPCBi ($i = 0, 1, \dots, n$) each of which stands for a logical processor thereof and contains state flags for managing its states, and a VMAL which stands for address list for VMBLOCKS. The hypervisor creates and manages these control blocks in main memory, scheduling logical processors using those LPCBs, managing their state flags, and defining Dflag (1) explained in section 3. This configuration is conventional and not new.

The relation between the hypervisor and the guest

OSs is similar to that between an OS and its application processes [9]. Contrary to this similarity, the scheduler of the hypervisor differs from a process scheduler of an OS in the following three points.

(1) An interrupt may be reported to a running logical processor directly, that is, by hardware and without intervention of the hypervisor.

(2) A logical processor in the running state will keep running immediately after it has accepted its interrupts directly.

(3) Conventionally if an interrupt request occurs and no logical processors of its target are running, it becomes a host interrupt, then, the hypervisor simulates it according to the hardware architecture and reports it to one of the target logical processors.

The differences (1) and (2) enhance the total system performance because of no intervention of the hypervisor, while the last difference (3) causes the guest I/O interrupt simulation and the mode change overhead stated in section 1, and reduces the system performance. We propose a new scheduling method to reduce those overheads in the following section.

3. New scheduling method

We propose a new scheduling method consisting of the following four methods:

- (1) adding a new self-wait state different from the conventional wait state,
- (2) detecting the I/O interrupts pending for the logical processors in the self-wait state by using a new instruction,
- (3) scheduling the logical processors in the self-wait state,
- (4) searching for workloads in an active wait state with two stages when workloads have been exhausted.

3.1. Self-wait state

If a running logical processor falls into the wait state, it is due to (1) the exhaustion of its workloads, or due to (2) the hypervisor commands to stop it. The operators do not issue the hypervisor commands to the logical processor during its production running, while they may issue them during its test running, which means a guest OS is tested in the logical processor.

In the former case, an OS in the logical processor usually falls into the system wait state, which will be waked up by an interrupt to it. In the latter case, the logical processor should be started also by some hypervisor commands. To distinguish these two kinds of wait state, we call the former the self-wait state.

Figure 4 shows conventional states (1), (2), (3),

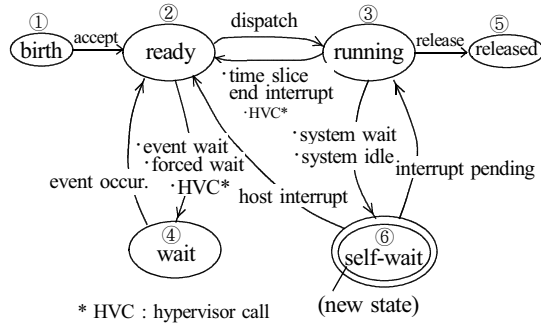


Fig. 4. Self-wait state of logical processors

(4), (5) and new state (6), that is, birth state (1), ready state (2), running state (3), released state (5), conventional wait state (4), and the new self-wait state (6). The logical processors in the self-wait state enter a self-wait queue, which is separated from a conventional wait queue, which consists of the logical processors in the conventional wait state (4). We present a method for detecting a VM's pending I/O interrupts by scanning this self-wait queue. This separation makes the scan quick. The interrupt delay is reduced because the scheduler of the hypervisor scans the self-wait queue ahead of the ready queue.

3.2. Conventional direct I/O execution method

Conventionally, I/O instructions issued by a guest OS are directly executed by hardware when they are issued to subchannels dedicated to the guest OS, and their interrupts are also directly executed during its running [2], [4]. This makes its performance close to that of the native machine. We premise that this direct execution method is already used in the real host machine. The conventional direct I/O execution method has been supported to the running logical processors. The I/O interrupts pending for VMs become host interrupts when all their target logical processors are in the wait state or in the ready state as stated in section 1. Our new method applies this direct execution method also to the self-wait logical processors though it may additionally delay the I/O interrupts to them.

Here, to clarify our method, we simply explain the conventional direct execution method having the following features,

(1) hardware logic recognizes the I/O interrupts pending for VMs without the intervention of the hypervisor,

(2) hardware logic judges whether or not a running logical processor is enabled for its pending I/O interrupts without the intervention of the hypervisor,

(3) hardware logic informs a running logical processor of its I/O interrupts when it is enabled for them without the intervention of the hypervisor.

To support the feature (1), we allocate I/O interrupt priorities exclusively to VMs [2], [3]. Figure 5 shows I/O interrupt request queues. It illustrates an example of the exclusive allocation of the I/O

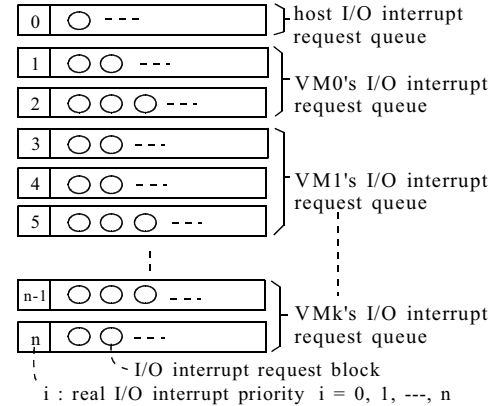


Fig. 5. I/O interrupt request queues (Conventional)

interrupt priorities 0, 1, 2, ..., n. It allocates priorities 1, 2 to VM0, priorities 3, 4, 5 to VM1, ..., priorities n-1, n to VMk, and allocates priority 0 to the hypervisor. An I/O interrupt request with priority (i) is enqueued into the request queue with priority (i). These I/O interrupt request queues are composed in real memory and managed by hardware, such as CPUs and I/O processors. With this queuing, hardware can recognize the I/O interrupt requests pending for VMs. For example, if an I/O interrupt request is enqueued into the queue with priority (i), and if the priority (i) is dedicated to VM1, the interrupt belongs to the VM1. As a result, hardware can recognize that the VM1 has the pending I/O interrupt requests.

This method does not require so many interrupt priorities, indeed, most OSs only use a few (probably two or three) priorities. This is because they will poll any I/O interrupts using some I/O instructions. These facts lead to this method which allocates a few interrupt priorities exclusively to VMs.

To support the feature (2) and (3), hardware has to have an interrupt control logic for a running VM which contains an I/O interrupt mask register and an I/O interrupt priority mask register. Several variations have been supported for those features, such as giving only one priority to each VM [3], or giving multiple priorities to each VM [2], or giving all the priorities

to each VM [4].

This conventional direct execution method can be applied only to the running logical processors. We apply it also to the self-wait logical processors.

3.3. Detecting I/O interrupts pending for self-wait logical processors

An I/O interrupt request occurs even when its target logical processor is in the wait state. Conventionally, it brings about a host interrupt. We present a method for suppressing this host interrupt. The I/O interrupt priorities are exclusively allocated to VMs as shown in Figure 5. We set the host masks of those exclusively allocated priorities always to be 0. This mask setting always suppresses the host I/O interrupts with those priorities. The pending I/O interrupts with those priorities are directly executed when one of its target logical processors are dispatched.

Conventionally, no scheduler dispatches the logical processors in the wait state or in the self-wait state. This is natural because the forced dispatching of the waiting logical processors will activate no activities and they will immediately fall into the wait state again, only increasing scheduling overhead.

After a logical processor falls into the self-wait state, its external system activities will generate a pending I/O interrupt or a pending timer interrupt or another interrupt pending for itself. When the logical processor is dispatched, those interrupts pending for itself can be directly executed by the hardware as long as the hardware can recognize those pending interrupts.

If the hypervisor is provided with a hardware function that can quickly detect the I/O interrupts pending for a VM, it can use the function to detect them. When detecting an interrupt pending for a self-wait logical processor, and it is enabled for that

interrupt, the hypervisor can dispatch it. Then, that interrupt request will be directly executed for it by the hardware.

Based on this consideration, we propose a new machine instruction called "Only Test Pending I/O Interrupts (OTPI)" shown in Figure 6, which detects an I/O interrupt pending with the priority i ($i = 0, 1, \dots, n$), which is an input parameter stored in a register R1, storing non-0 in a register R2 when detected, or 0 otherwise as a result. This instruction can be defined in any hardware architecture that has the I/O interrupt priorities $0, 1, \dots, n$ into which hardware enqueues the I/O interrupt requests.

Users allocate real I/O interrupt priorities exclusively to VMs using hypervisor commands. The hypervisor composes logical control blocks shown in Figure 3, which contains a Dflag (①), which indicates real interrupt priorities dedicated to the VM. By accessing it, the hypervisor can get those priorities, and can use this instruction with them, and can determine whether the VM has any pending I/O interrupts with those priorities.

The hardware usually has I/O interrupt request queues shown in Figure 5. This instruction is very simple and only checks the queue with the designated priority, and much faster than the I/O interrupt execution. This is because it only checks the designated queue and does not dequeue any I/O interrupt request element, while the I/O interrupt execution dequeues its request element and carries out the subsequent hardware interrupt procedures. Thus, the hypervisor can quickly determine whether a waiting logical processor has an I/O interrupt pending with its dedicated priorities without any host I/O interrupt for them.

We say that a logical processor is in the "self-wait-ready state," when it is in the self-wait state, at the same time, it has an I/O interrupt pending for itself, and it is enabled for that interrupt.

Conventionally, IBM provides an instruction called "Test Pending Zone Interrupt (TPZI)" [3], which checks the I/O interrupts of the designated guest, selectively can complete here, or continuously dequeues the I/O interrupt requests elements, and informs the hypervisor of the interrupt information. We think that this TPZI instruction is probably used to reap the guest I/O interrupt requests in the hypervisor of IBM's virtual machine system. This instruction seems to correspond to "Test Pending Interrupt (TPI)" instruction [12], [13] that native OSs use to check the pending I/O interrupts in the hardware queue and to dequeue (i.e. reap) the I/O interrupt request elements in the queue. Our OTPI instruction only tests the I/O interrupts pending in the queue and does not dequeue any request elements, therefore, much faster than those instructions. We do

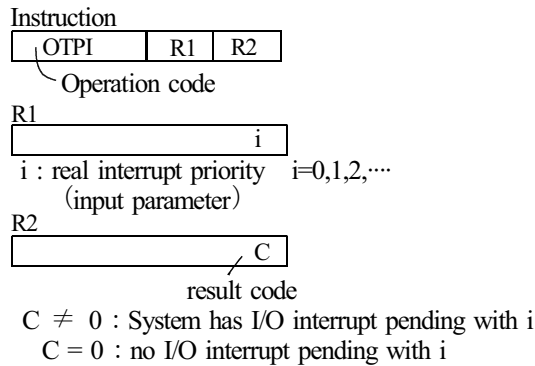


Fig. 6. Only Test Pending I/O Interrupt (OTPI) instruction

not know in detail how those other similar instructions are used in IBM's OSs or in the hypervisor of its virtual machine system.

3.4. Scheduling logical processors in self-wait queue

Figure 7 shows a new scheduling method using the self-wait queue ((a)). First, the scheduler checks the

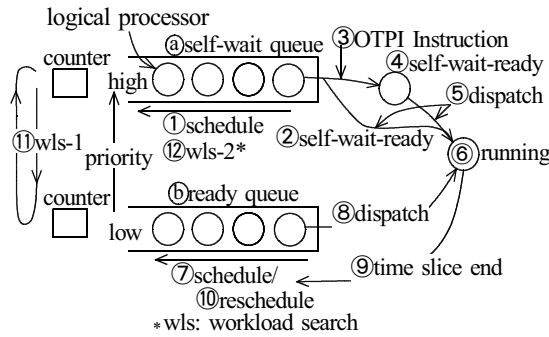


Fig. 7. Scheduling logical processors in self-wait queue

top entry of the logical processors in the self-wait queue (①). If it (i.e. the top entry) is in the self-wait-ready state (②), the scheduler dispatches it (⑤), otherwise checks whether it has an I/O interrupt pending with its dedicated priorities by using OTPI instruction (③). If it has an I/O interrupt pending for itself and if it is enabled for that interrupt, the scheduler sets it self-wait-ready ((4)), dispatches it (⑤), and sets it running (⑥).

Immediately after this dispatching, the real processor goes into vm mode, and directly executes the I/O interrupt pending for the running logical processor. If it (the top entry) has no pending I/O interrupts, the scheduler checks the next entry (i.e. a next logical processor) in the self-wait queue.

Second, when finding no dispatchable logical processors in the self-wait queue, the scheduler goes to the ready queue ((b)), and dispatches a ready logical processor if any (⑧). Thus the scheduler prefers the self-wait queue to the ready queue to shorten the additional delay of the I/O interrupts. Even in this ready queue, when finding no dispatchable entry, the scheduler goes to the workload search (⑪, ⑫), (Refer to 3.5).

The scheduler tries to keep running the logical processor in its time slice until it falls into the time slice end or the self-wait state. When the running logical processor falls into its time slice end (⑨), the scheduler reschedules it (⑩) according to its CPU service quantity, and gives higher priority to the logical processors with low CPU service quantities.

The additional delay of the guest I/O interrupts may occur because of no host interrupts pending for VMs. In order to shorten the additional delay, the scheduler gives the precedence to the self-wait queue over the ready queue, and checks the self-wait queue at every host event such as the host time slice end interrupts.

Conventionally a system has the interrupt delay due to the scheduling of its processes. That is, when the system has the processes with the same priority, and its process P0 is in the wait state, and an I/O interrupt to the process P0 makes it ready, and it is enqueued in its ready queue, and if there are k (≥ 1) ready processes before the process P0, then its reporting to the process P0 is delayed at most $k \cdot q$, where q is the time slice. Even if the system has several CPUs or many process priorities, a similar delay occurs.

To this conventional delay, our method adds at most one time slice because the host timer interrupt for the time slice end makes the scheduler quickly check the self-wait queue and dispatch the logical processor if it has an I/O interrupt pending for itself. The time slice is ordinary 1 ms to 10 ms, and thus usually causing no problem.

This additional delay may cause a timing problem for a VM that runs hard real time processes. In that case we can dedicate several processors to the VM in the SMP system [11], where they always run the VM in the vm mode and never fall into the self-wait state, and all its interrupts are executed directly with no additional scheduling delay.

3.5. Searching for workloads in active wait state with two stages

When finding no logical processors dispatchable in the current real processor, the scheduler will conventionally make it fall into the system wait state or actively search its workloads [10]. These conventional methods will find out its workloads by accepting host interrupts pending for guest VMs.

Instead of these methods, we present the method which causes no host I/O interrupts pending for VMs, and provides a workload search with two stages (see Figure 7), where the hypervisor has to accept the host timer interrupts and has to actively detect the I/O interrupts pending for the self-wait logical processors.

Figure 7 shows the workload search with two stages one of which is wls-1 (⑪) and the other is wls-2 (⑫). The first stage (wls-1) opens host interrupt masks for non-dedicated I/O interrupt priorities, and checks two counters, the self-wait queue counter and the ready queue counter.

Some waiting logical processors may come into the ready queue by host interrupts which may be host timer interrupts or may be host I/O interrupts with shared priorities. When finding the ready queue counter non-zero, the hypervisor exits from this stage (wls-1) and goes to the scheduler, when finding the ready queue counter zero and the self-wait queue counter non-zero, it exits from this stage, and goes to the second stage (wls-2), and when finding both counters zero, it loops between them.

In the second stage (wls-2) the hypervisor scans the self-wait queue, and checks if each logical processor has I/O interrupts pending for itself. The hypervisor checks the self-wait logical processor (we call this A) by issuing an OTPI instruction whose operand is an I/O interrupt priority dedicated to the logical processor A, and sets its self-wait-ready flag if it has the I/O interrupt pending for itself and is enabled for that interrupt.

In that case the hypervisor sets the self-wait-ready flags of all the self-wait logical processors that are brothers of the logical processor A, that is, belong to the same VM. This is because there may be several I/O interrupts pending for the VM in the interrupt request queues with its dedicated priorities (Refer to Figure 5).

The hypervisor can easily find dedicated I/O priorities by seeing the Dflag (①) of the logical processor as shown in Figure 3, where priorities $i, i+1, \dots, i+j$ ($j \geq 0$) are dedicated to a VM (represented as VMBLOCK), and to logical processors $0, 1, \dots, n$ (as LPCB0, LPCB1, $\dots$, LPCBn, respectively). It can also easily find all the logical processors belonging to the same VM by seeing its pointers (②) in Figure 3. Thus when finding at least one self-wait-ready logical processor, the hypervisor exits from the stage 2 workload search and goes to the scheduler to dispatch it, when finding no such logical processors, it returns to the first stage (wls-1).

The pending I/O interrupts may occur asynchronously to the real processors. The hypervisor has to get the logical processor's most current state. Therefore, it has to issue the OTPI instruction each time it checks if the logical processor has a pending I/O interrupt. It is not sufficient for the hypervisor to read all the pending mask bits only once of all self-wait logical processors because the values of their mask bits may vary momentarily. Moreover, the hypervisor should schedule them in the order of their self-wait states and their pending I/O interrupts.

4. Related art and architecture

Our proposed method schedules and dispatches the self-wait logical processors. A conventional OS's

scheduler does not dispatch the waiting processes, usually accepting interrupts for them, changing their states from the wait to the ready, after that, dispatching only the ready state processes, thus having no methods for scheduling and dispatching the waiting processes directly.

Active wait method is supported conventionally [10]. It opens all the interrupt masks, causing any kind of interrupts, resulting in the increase of their simulation overhead in the VM environment. Instead of this, our active wait method disables for the I/O interrupts pending for VMs with dedicated interrupt priorities, detecting them without host interrupts, responsive to the detection, scheduling and dispatching the VMs.

Conventionally, Denali [7] has the lazy interrupt processing and Xen [8] has the batch processing for interrupts. In both methods the hypervisor accepts I/O interrupts pending for VMs, translating them into simple events, and reporting them at the scheduling of the VM. This is the I/O simulation and will cause high simulation overhead.

Contrary to this, in our method the hypervisor never accepts host I/O interrupts pending for a VM with dedicated interrupt priorities. The hypervisor detects those I/O interrupts pending for a waiting logical processor of a VM by using the new OTPI instruction, which can quickly detect them. After detecting them and checking if it is enabled for them, the hypervisor schedules it, then, they are directly executed in the vm mode hardware, thus causing no I/O simulation overhead.

Our method also has the conventional scheduling delay, and moreover may have the additional interrupt delay which is at most one scheduling time quantum due to the masking of the host I/O interrupts as described in section 3.4. We premise that our virtual machine system has the small number of VMs, that is, having at most ten VMs in the system. Therefore, these scheduling delays cause no problem because of the small time slice such as 1ms -10ms. The number of VMs is small because the guest OSs have almost the full use of the hardware and software functionalities and can run in near native performance, and will have the scalability to the number of their own processes.

Our method premises that the hardware architecture has a feature which supports direct I/O execution for VMs in their *running* states. It is widely supported in mainframe architectures [2], [4].

5. Implementation and performance evaluation

5.1. Implementation

As for hardware we have implemented the OTPI instruction, which detects I/O interrupts pending of the designated interrupt priority (See Figure 6). We used a hardware microcode program and two general purpose registers. This was very easy because the basic I/O interrupt queues were supported natively in hardware, only requiring additional control storage for its microcode.

As for software we have implemented the self-wait state, the mask setting to disable for the host I/O interrupts pending for VMs, the method for scheduling the self-wait logical processors, and the workload searching with two stages.

We have evaluated our scheduling method consisting of these proposed methods by measuring the performance for several workloads. Table 1 summarizes the performance comparison between the conventional host interrupt method and our proposed method. We describe the measurement method and the performance data in the following sections.

5.2. Measurement method

We address the users who daily operate multiple VMs concurrently under a single real host SMP. Several users have very large size systems consisting of several SMPs each of which has many (up to 128) processors sharing very large size (up to 2048GB) main memory and concurrently runs several (up to tens of) VMs. On the other hand, many users have medium size systems each of which will be a 2 to 4-way SMP system and runs 2 to 4 VMs concurrently.

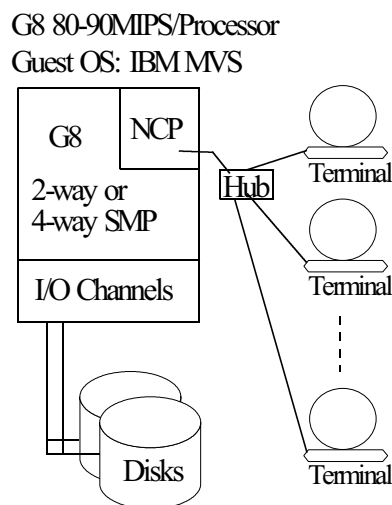


Fig. 8. Measurement Environment

Because our VMs can use almost the full functions of hardware and software, the number of VMs concurrently running is not so large, that is, up to ten. This is very different from the Denali VMs [7] and from the Xen VMs [8].

Considering that our virtual machine system has the small number of VMs running concurrently, we need not evaluate the scalability to the great number of VMs concurrently running. The scalability should be evaluated to the number of processes in each of the guest OSs. Here, the most important is the performance of each VM to the native machine.

We use the measurement environment shown in Figure 8, using a 4-way SMP, which consists of 4 processors sharing main memory, and a 2-way SMP, which consists of 2 processors sharing main memory.

Each processor has a 6.2ns machine cycle. This SMP has almost the same architecture as IBM System/390 [12]. The guest OS is the MVS, which is the OS of the IBM mainframe.

The number of logical processors of a VM is the same as that of the real host SMP. This is because the VM should have as many concurrently running logical processors as possible to improve its performance.

That is, when a real host is a 4-way SMP, we run 2 to 4 VMs, each of which is 4-way, that is, has 4 logical processors. When a 2-way SMP host, we run 2 VMs, each of which is 2-way, that is, has 2 logical processors.

The measurement of one case takes 1 to 3 hours including preparations, and measured workloads are I/O heavy jobs which seem to have the size and attributes comparable with real workloads. We use a performance monitor built in the guest OSs, summarizing the results in Table 1, explaining each case as follows.

Case 1 (#1 in Table 1) runs a 4-way SMP as a real host, in which running 2 VMs concurrently, each of which is 4-way, that is, has 4 logical processors. Its workloads are the Time Sharing System (TSS) jobs with 100 active terminals, and each VM has 50 terminals. We use a terminal simulator to generate the transactions each of which consists of a series of edit and its subcommands. We run the total system for about 15 minutes, getting performance data for two minutes (i.e. measurement interval in Table 1) of steady state during that running interval.

Table 1. Performance comparison of conventional method (a) with proposed method (b)

#	Workload	Item	Conventional method (a)	Proposed method (b)	Native mode (c)	b-a	(b-a)/a (%)	a/c (%)	b/c (%)	Comments
1	TSS 100 terminals	ETR ^{*1} (Ended/sec)	51.60	53.57	53.66	1.97	3.82	96.16	99.83	Host:4way 2 VMs each VM:4way $\rho^* = 26\%$
		Simul. ovh. ^{*3} (%)	3.21	1.85	—	-1.36	-42.37			
		Sched. ovh. ^{*4} (%)	15.38	1.77	—	-13.61	-88.49			
		Interval ^{*5} (sec)	120.00	120.00	120.00	0.00				
2	I/O_heavy	ETR ^{*2} (EXCPs/sec)	12.65	16.34	16.90	3.69	29.15	74.87	96.69	Host:2way 2 VMs each VM:2way $\rho^* = 49\%$ (estimated from VM mode)
		Simul. ovh. ^{*3} (%)	12.95	7.43	—	-5.52	-42.63			
		Sched. ovh. ^{*4} (%)	20.17	5.79	—	-14.38	-71.29			
		Interval ^{*5} (sec)	300.00	300.00	300.00	0.00				
3	I/O_heavy	ETR ^{*2} (EXCPs/sec)	13.30	16.31	16.70	3.01	22.63	79.64	97.66	Host:4way 2 VMs each VM:4way $\rho^* = 25\%$
		Simul. ovh. ^{*3} (%)	13.86	8.95	—	-4.91	-35.43			
		Sched. ovh. ^{*4} (%)	42.96	8.47	—	-34.49	-80.28			
		Interval ^{*5} (sec)	300.00	300.00	300.00	0.00				

*: ρ : Native CPU utilization

Host Machine Cycle : 6.2ns

*1: ETR: External Throughput Ratio: Transactions ended / Real time one sec.

*2: ETR: External Throughput Ratio: Execute Channel Programs issued / Real time one sec.

*3: CPU utilization for simulating system wait inst.(Load PSW wait) and host timer interrupts that should be reported to guests

*4: CPU utilization for scheduling logical processors

*5: Measurement interval

Case 2 (#2 in Table 1) runs a 2-way SMP as a real host, in which running 2 VMs concurrently, each of which is 2-way, that is, has 2 logical processors. Its workloads are I/O heavy jobs issuing many Execute Channel Program (EXCP) macro-instructions, which are I/O macro-instructions, very frequently. We get the performance data for 5 minutes (i.e. measurement interval in Table 1) of steady state during that running interval.

Case 3 (#3 in Table 1) runs a 4-way SMP as a real host, in which running 2 VMs concurrently, each of which is 4-way, that is, has 4 logical processors. The other attributes are the same as the case 2.

5.3. Performance data

(1) Simulation Overhead (CPU utilization)

The simulation overhead belongs to each VM. It reduces the total system performance because it decreases of the CPU times for other VMs. Table 1 shows that the simulation overhead remarkably decreases compared with that of the conventional

method, that is, in CPU utilization, 3.21% to 1.85% in case 1, 12.95% to 7.43% in case 2, and 13.86% to 8.95% in case 3. This data means that our proposed method reduces the number of host I/O interrupts pending for the guest OSs, and the self-wait logical processors directly execute their own I/O interrupts at the time they have been dispatched.

(2) Scheduling Overhead (CPU utilization)

The scheduling overhead is the CPU utilization for scheduling logical processors, and if it is high, it reduces the total system performance. The overhead is remarkably reduced compared with that of the conventional method as Table 1 shows that 15.38% to 1.77% in case 1, 20.17% to 5.79% in case 2, and 42.96% to 8.47%. The high scheduling overhead of the conventional method is due to the high frequency of the host I/O interrupts pending for the guest OSs and its poor workload searching.

(3) External Throughput Ratio (ETR)

The external throughput ratio (ETR) is the number of transactions processed in case 1, or the number of EXCP (Execute Channel Program) macro-instructions issued in cases 2 and 3 in a real time second. This is

the performance from the point of user view.

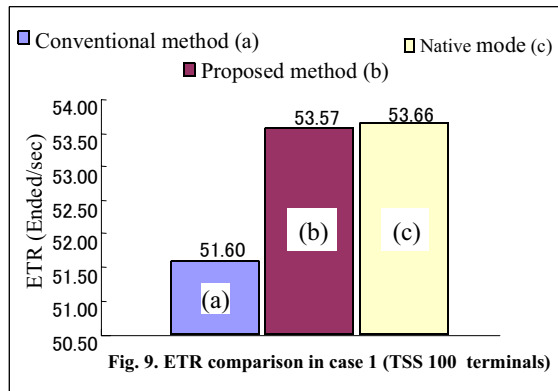


Fig. 9. ETR comparison in case 1 (TSS 100 terminals)

As for case 1 (TSS 100 terminals) Figure 9 shows a visible ETR improvement from 51.60 to 53.57.

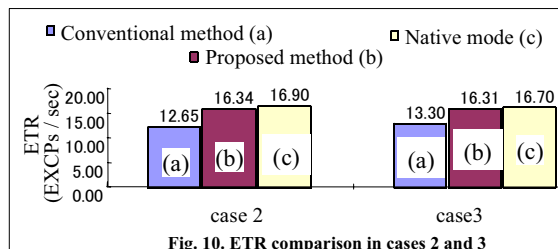


Fig. 10. ETR comparison in cases 2 and 3

As for cases 2 and 3 (I/O heavy jobs) Figure 10 shows a clear ETR improvement from 12.65 to 16.34 in case 2, and from 13.30 to 16.31 in case 3, additionally showing the ETR improvement reaching the near native performance.

These ETR improvements are almost due to the decrease of the I/O interrupt simulation for VMs and that of scheduling overhead.

6. Conclusions

We have presented a new method for scheduling logical processors in the self-wait state ahead of those in the ready state, and for searching workloads using two stages which scan the self-wait logical processors.

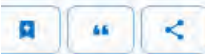
Having implemented this new method, we have evaluated it by measuring the performance for some workloads, and have confirmed their improvements by the performance data showing the decrease of the I/O interrupt simulation overhead and that of the

scheduling overhead. Moreover, we have confirmed the ETR performance improvement reaching the near native even for the I/O heavy workloads.

7. References

- [1] IBM, "Logical Partitioning", <http://www-1.ibm.com/servers/eserver/series/lpar/>.
- [2] Umeno, H., et al., "I/O execution method for a virtual machine system and system therefor", Hitachi, US 5109489, Priority Jan. 18, 1984.
- [3] Bean, George H., et al., "Logical Resource Partitioning of a Data Processing System", IBM, USP4843541, July 29, 1987 Filed.
- [4] Chou, Norman Cho-Chun, et al., "CPU expansive gradation of I/O interruption subclass recognition", US 752149, IBM, Filed Aug. 29, 1991.
- [5] Sugerman, J., et al., "Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor", VMware, Inc., Proc. of the 2001 USENIX Annual Tech. Conf., Boston, Ma., USA, June 25-30, 2001.
- [6] Scott W. Divine, et al., "Virtualization System Including a Virtual Machine Monitor for a Computer with a Segmented Architecture", US6397242B1, VMware Inc., Filed Oct. 26, 1998.
- [7] Whitaker, A., Shaw, M., Gribble, Steven D., "Scale and Performance in the Denali Isolation Kernel", In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI 2002)*, ACM Operating Systems Review, Winter 2002 Special Issue, Boston, MA, USA, Dec. 2002, pp. 195-210.
- [8] Barham, P., Dragovic, B., Fraser, K., et al., "Xen and the Art of Virtualization", SOSP'03, October 19-22, 2003, BoltonLanding, NY, USA, pp. 164-177.
- [9] Stallings, W., "Operating Systems: Internal and Design Principles Fourth Edition", Prentice Hall (2000).
- [10] Blandy, G. O., et al., "Active wait", US4631674, IBM, Filed Feb. 5, 1985.
- [11] Umeno, H., et al., "Development of Methods for Reducing the Spins of Guest Multiprocessors", Trans. IPS Japan, Vol. 36, No. 3, pp. 681-696 (March, 1995).
- [12] IBM, "ESA/390 principles of operation", SA22-7201-08.
- [13] IBM, "z/Architecture Principles of Operation", SA22-7832.





Proceedings of the 29th Annual International Computer Software and Applications Conference : COMPSAC 2005 : July 26-28, 2005, Edinburgh, Scotland

[Show less ^](#)

Authors: [COMPSAC](#), [IEEE Computer Society](#)

 **Print Book**, English, ©2005

Edition: [View all formats and editions](#)

Publisher: IEEE Computer Society, Los Alamitos, Calif., ©2005

Genre:	Conference papers and proceedings
Physical Description:	xix, 565 pages : illustrations ; 28 cm
ISBN:	9780769524139, 0769524133
OCLC Number / Unique Identifier:	61722758
Subjects:	Computer networks Computer networks Congresses Computer programming Computer programming... Show more v
More Information:	IEEE Xplore

[Show less information ^](#)

Find a Copy at a Library Filter by: [Any format v](#) [Any edition v](#) [Distance within 200+ mi v](#)

 **2 editions in 19 libraries**

Screen capture completed by JL Mullins from WorldCat on November 8, 2022

Academic



[North Carolina State University](#)

NC State University Libraries

 17 miles from your current location.

2 Broughton Dr., Campus Box 7111
Raleigh, NC 27695-7111
United States

[Get Directions](#) 



Favorite Library

[Borrow](#) 

Proceedings of the 29th Annual International Computer Software and Applications Conference : COMPSAC 2005 : July 26-28, 2005, Edinburgh, Scotland - NC State University Libraries Catalog



[\(https://www.lib.ncsu.edu/\)](https://www.lib.ncsu.edu/)

Proceedings of the 29th Annual International Computer Software and Applications Conference : COMPSAC 2005 : July 26-28, 2005, Edinburgh, Scotland

[Start over](#) [Back to Search](#)

sponsored by IEEE Computer Society.
COMPSAC (29th : 2005 : Edinburgh, Scotland)
Los Alamitos, Calif. : IEEE Computer Society, [2005] / Book /
Print



Where to find it

Hunt Library [\(https://www.lib.ncsu.edu/huntlibrary/\)](https://www.lib.ncsu.edu/huntlibrary/) — bookBot

Call Number

QA76.6 .C6295 2005 V.1

[\(Browse Shelf\)](#)

Status

Available upon request

[Request Item](#)

Call Number

QA76.6 .C6295 2005 V.2

[\(Browse Shelf\)](#)

Status

Available upon request

[Request Item](#)

Authors, etc.

- [COMPSAC \(29th : 2005 : Edinburgh, Scotland\) \(https://catalog.lib.ncsu.edu/?q=%22COMPSAC+%2829th+%3A+2005+%3A+Edinburgh%2C+Scotland%29%22&search_field=author\)](#)
- [IEEE Computer Society \(https://catalog.lib.ncsu.edu/?q=%22IEEE+Computer+Society%22&search_field=author\)](#)

Summary

COMPSAC is a maior international forum for researchers. practitioners.

YES (HTTPS://LIBRARYH3LP.COM/CHAT/NCU1866647?SKIN=29467&REFERER=HTTPS%3A%2F%2FNCU1866647)

TALOG@CHAT.LIBRARYH3LP.COM?SKIN=29467&REFERER=HTTPS%3A%2F%2FNCU1866647

managers, and policy makers to discuss the state of the art, new advances, and future trends in software technologies and practices. This year's theme is "High Assurance Software Systems". COMPSAC 2005 explores a broad and diverse range of both technical and non-technical topics, from basic methodology and software process design to practical software concerns. There is a second volume of Workshop papers and fast abstracts. The fast abstracts provide authors an opportunity to report on-going work, describe practical experience, introduce new ideas to provoke further validation, or state positions on controversial issues.

Content provided by Syndetic Solutions, Inc. [Terms of Use](#)

Contents

[v. 1.] Proceeding papers – [v. 2.] Workshop papers and fast abstracts.

Find similar items

[Browse shelf](#)

By Topic

- [Computer programming](#) > [Congresses](#)
- [Programming languages \(Electronic computers\)](#) > [Congresses](#)
- [Computer networks](#) > [Congresses](#)

Other details

Variant Title:

Cover title: The Twenty-Ninth Annual International Computer Software & Applications Conference.

Published:

Los Alamitos, Calif. : IEEE Computer Society, [2005]
©2005

Language:

English

Notes:

"IEEE Computer Society Order Number P2413"—T.p. verso.
Includes bibliographical reference and index.

Description:

2 volumes : illustrations ; 28 cm

Genre or Form:

[Congresses](#)
[Conference papers and proceedings](#)

ISBN:

0769524133

OCLC Number:

61722758

YES (HTTPS://LIBRARYH3LP.COM/CHAT/NCS
SKIN=29467&REFERER=HTTPS%3A%2F%2FCATALOG.LIB.NCSU.EDU%2FCATALOG%2FNCSU1866647)

View MARC

ISBN:

0769524133



MARC for NCSU1866647

000 01835cam a2200409Ia 4500
003 OCoLC
005 20051012031732.0
008 050829t20052005caua b 101 0 eng d
020 |a0769524133
035 |a(OCoLC)61722758
040 |aCAI|beng|cCAI
049 |aNRCC|xtmc
090 |aQA76.6|b.C6295 2005
1112 |aCOMPSAC|n(29th :|d2005 :|cEdinburgh, Scotland)
24510 |aProceedings of the 29th Annual International Computer Software and Applications Conference :|bCOMPSAC 2005 : July 26-28, 2005, Edinburgh, Scotland /|csponsored by IEEE Computer Society.
24614 |aThe Twenty-Ninth Annual International Computer Software & Applications Conference.
24630 |a29th Annual International Computer Software & Applications Conference.
24630 |aCOMPSAC 2005.
2463 |aCOMPSAC05
264 1 |aLos Alamitos, Calif. :|bIEEE Computer Society,|c[2005]
264 4 |c02005
300 |a2 volumes :|billustrations ;|c28 cm
336 |atext|btxt|2rdacontent
337 |aunmediated|bn|2rdamedia
338 |avolume|bnc|2rdacarrier
500 |a"IEEE Computer Society Order Number P2413"--T.p. verso.
504 |aIncludes bibliographical reference and index.
5050 |a[v. 1.] Proceeding papers -- [v. 2.] Workshop papers and fast abstracts.
596 |a14

650 0 |aComputer programming|vCongresses.|0http://id.loc.gov/authorities/subjects/sh2007004295
650 0 |aProgramming languages (Electronic computers)|vCongresses.|0http://id.loc.gov/authorities/subjects/sh2008108876
650 0 |aComputer networks|vCongresses.|0http://id.loc.gov/authorities/subjects/sh2008101222
655 7 |aConference papers and proceedings.|2lcgft
710 2 |aIEEE Computer Society.|0http://id.loc.gov/authorities/names/n79043202
918 |a1866647
994 |aC0|bNRC
917 |au1866647

From: [Beth Ashmore](#)
To: [Mia Partlow](#)
Cc: [JAMES L. MULLINS](#); [Lynn Whittenberger](#); [Thomas Jones](#)
Subject: Re: NC State Library Article Request
Date: Monday, November 7, 2022 3:10:47 PM
Attachments: [image002.png](#)

----- **External Email:** Use caution with attachments, links, or sharing data -----

Hi Mia and Dr. Mullins,

I agree that the 005 is an appropriate substitute for a dated accession stamp. 10/12/2005 matches the cataloged date on our bibliographic control screen in our catalog. Unfortunately, we don't have any order record information in our system that would tell us more specifically when the item was received prior to cataloging.

Lynn, if there is another place to check for this info, let me know.

Hope that helps!

-Beth

On Mon, Nov 7, 2022 at 2:57 PM Mia Partlow <mpduffy@ncsu.edu> wrote:

Hi Dr. Mullins,

I have CC'd two colleagues, Lynn Whittenberger and Beth Ashmore, who can help answer your question about the MARC record.

Lynn and Beth: Dr. Mullins is looking at [this record](#). He originally requested a scan of the accession date stamp, which the physical item does not have, and is wondering if the 005 field is an appropriate substitute. (Please correct me if I've got anything wrong, Dr. Mullins!)

Cheers,
Mia

On Mon, Nov 7, 2022 at 1:04 PM JAMES L. MULLINS <jmullins@purdue.edu> wrote:

Thomas, thank you so much for filling this request so quickly, I greatly appreciate it.

Since there is not a stamped date of receipt, only the ownership stamp, however, I can use the MARC Format 005 field that reads

005 20051012031732.0

I believe the 005 MARC Field indicates when it was received/cataloged on October 12, 2005, is there a person you could direct me to in the cataloging department who could

document that I am reading the OO5 Field correctly?

Thank you,

Jim

James I. Mullins, PhD

Dean Emeritus of Libraries &

Norton Professor Emeritus

Purdue University

205 St. Cuthbert

Williamsburg, VA 23188

Ph. 765 479 4956

jmullins@purdue.edu

From: Thomas Jones <tljones8@ncsu.edu>
Sent: Monday, November 7, 2022 9:26 AM
To: JAMES L. MULLINS <jmullins@purdue.edu>
Cc: Mia Partlow <mpduffy@ncsu.edu>
Subject: NC State Library Article Request

---- **External Email:** Use caution with attachments, links, or sharing data ----

Hi Dr. Mullins,

Here is the requested article and accompanying proceedings information you requested from NC State Libraries. I was able to scan everything you asked for other than an accession date stamp. Unfortunately, most of our materials do not have those. Please let us know if you need anything else or if you have any questions.

Thank you,

Thomas Jones

--

Thomas L Jones

University Library Specialist, Access Services

North Carolina State University Libraries

Phone: 919-513-3308

he / him

--

Mia Partlow (she/her)

Resource Sharing Librarian

NC State University Libraries

919-515-3407

mia_partlow@ncsu.edu

--

Beth Ashmore

Associate Head, Acquisitions and Discovery (Serials)

North Carolina State University Libraries

beashmor@ncsu.edu

919-513-3354

(Pronouns: she, her, hers)



I may email you outside your working hours - please don't feel pressure to respond until your working hours.

All electronic mail messages in connection with State business which are sent to or received by this account are subject to the NC Public Records Law and may be disclosed to third parties.