Proceedings of the Twentieth Annual Hawaii International Conference on System Sciences, 1987.

A CLARIFICATION OF THE DYNAMIC/STATIC INTERFACE

Stephen W. Melvin
Yale N. Patt

Computer Science Division
University of California, Berkeley
Berkeley, CA 94720

ABSTRACT

Much has been written about the interface between hardware and software and the tradeoffs associated with it. In this paper, we show that there exists a separate interface which is commonly confused with the hardware/software interface (HSI). This is the dynamic/static interface (DSI), which defines the boundary between interpretation and translation. In this paper we show that the DSI is separate from the HSI and demonstrate the usefulness of this concept.

1. INTRODUCTION

A computer is a multilevel system with problems at the top and circuits at the bottom. In between are levels, or interfaces, which define sets of data structures and the operations allowed on them. For example, high level languages, machine architectures and microarchitectures are all interfaces.

A widely discussed interface is the hardware/software interface, or HSI, which defines the boundary between "hardware" and "software". Another interface, not as widely discussed but equally if not more important is the dynamic/static interface, or DSI, which defines the boundary between translation and interpretation. We will show that the DSI and the HSI are often confused, but are actually separate interfaces. Furthermore, we will show that by clarifying the concepts of the DSI and the HSI, different approaches to computer architecture can be put into a new perspective that we believe is useful. We will also discuss two other interfaces, the single cycle interface, or SCI, and the builder/user interface, or BUI.

This paper is divided into four sections. In this section we define the DSI, HSI, BUI and SCI and discuss general tradeoffs associated with the DSI. Section 2 provides some historical background on the DSI concept. Section 3 discusses various examples in terms of their interface configurations. Section 4 concludes with some final remarks.

1.1. DEFINITIONS

The dynamic/static interface arises from the fact that problem solutions typically undergo two stages. In the first stage, *translation*, the specification of the problem is changed from one form into another (i.e., a new problem specification is created at a lower interface). In the second stage, *interpretation*, the problem specification is executed, possibly using input data that is not part of the specification, and results are generated. The DSI is the interface between translation and interpretation.

For conventional machines running compiled languages, the DSI is at the machine architecture level. The high level language is translated into machine language, which then gets interpreted by the hardware. Because this case is so common, the DSI tends to be implicitly placed at the machine architecture level. Often a discussion of "hardware vs. software" will include points that are really related to the issue of static vs. dynamic.

Now suppose we have a program that interprets a high level language. In this case, the DSI is above the machine architecture level. The interpreter is clearly software, so the DSI must be different from the hardware/software interface. Conversely, hardware can be built which translates a program from one interface to another. In this case, the DSI is below the hardware/software interface.

The specific definition of the HSI presents some difficulties. The problem is that the question of what is "hardware" and what is "software" doesn't have a simple answer. The extremes are easy to identify, but the distinction is less clear in between.

We believe that the crucial element in the way most people interpret *software* and *hardware* is the question of alterability. In other words, how easy it is to alter a particular interface is related to how "soft" the interface

is. In this paper, we will use the following definition for the HSI: the highest interface that is not *dynamically alterable*. That is, if an interface cannot be changed while the computer is operating, everything below will be called hardware. Note that we could also have used a test of static alterability (i.e., can the interface be changed without physically modifying the computer?). Perhaps the HSI should be broken down into the hardware/firmware interface and the firmware/software interface and the tests of static and dynamic alterability should be applied to each interface respectively.

Under the dynamic alterability definition of the HSI, the microcode of most machines, even though it may be stored in read/write memory, is hardware because it cannot be changed without halting the processor. Even the microcode of the IBM System/370 model 145, which is stored in main memory, would be considered hardware under this definition because the memory region containing the microcode is protected and cannot be changed without rebooting [17]. However, the microcode of machines such as the B1700 [21] and the QM-1 [28] is software under this definition because it can be modified while the processor is running.

There are also ways to define hardware and software that have nothing to do with alterability. Patt and Ahlstrom [22] argue that microcode should be considered hardware if it is provided by the manufacturer and software if it is written by the user. They have called the hardware/software interface what might be more appropriately called the builder/user interface, or BUI. This interface defines the boundary between what the builder provides and what the user has access to. This definition of hardware and software, however, was rejected by a Federal District Judge in the recent Intel/NEC case. The microcode of the 8086 was ruled to be software even though it is not visible to the user.

Another interface that is useful to discuss is the single cycle interface, or SCI. This is the lowest interface of a machine; it is generally interpreted by combinational logic only. In the case of a pipelined implementation, there may be sequential logic below the SCI, but a particular piece of the data path is not used more than once for each instruction at the SCI level. For conventional machines, the SCI is the microarchitecture (i.e., the microword format, the definitions of the microorders and the definitions of the internal registers).

1.2. DSI TRADEOFFS

There are many tradeoffs associated with the DSI. We are mainly interested in how *movement* of the DSI affects performance, but it is necessary to understand that performance is variable even with a fixed DSI (i.e., translation time can be traded off for interpretation time). The translator can be very complex, performing many optimizations and making the interpretation phase faster, or it can be very simple, making the interpretation phase slower. In the extreme case, if a problem needs no input data, the translator could do what amounts to executing the entire program and then generate just the statements that print the result.

Movement of the DSI has different implications. If the DSI is at a suitably matched high semantic level, then the translation process would be simple but interpretation would be complex and might require several levels. In this case, the highest level interpreter would be executing instructions which would themselves be interpreted by a lower level interpreter.

On the other hand, if the DSI is at a very low semantic level, for example at the SCI, then there is no interpretation involved except by the circuits below the SCI. Translating to this interface involves several issues. Depending on the complexity of the SCI, translating near optimally may be very difficult. Translating to a simple SCI is easier, but performance may be related to the complexity of the SCI (e.g., how many things can be done in one cycle). Note that the problems associated with translating to the SCI are different from the problems associated with writing an interpreter which executes above the SCI. Thus, it is not always straightforward to compare the compilation to microcode with the creation of a macrocode interpreter in microcode.

There are also bandwidth tradeoffs. Lowering the DSI increases the run-time bandwidth to the highest level interpreter. Raising the DSI would lower the bandwidth required to this interpreter, and since the lower level interpreters are generally smaller, this could be advantageous.

2. HISTORICAL BACKGROUND

The concept of hardware and software and the concept of interpretation and translation have both been around for many years. However, the connection of the two in a coherent manner has not. In this section, we will provide a brief background on how others have viewed the DSI and the HSI and to what extent they have connected the two.

People have long recognized the two-phase nature of the execution of most programs. Hoevel [16] addresses the DSI directly and argues that it should be above the SCI but below the high level language. Flynn [8],[9] also distinguishes the DSI. These papers, however, don't discuss the DSI in connection with the HSI.

Myers, in chapter 3 of [20], compares some basic approaches to computer architecture. He distinguishes five approaches: traditional, language-directed, type A HLL machines, type B HLL machines and type C HLL machines. The main feature separating these ap-

proaches is the level of the DSI, not the level of the HSI. The discussion centers on the translation and interpretation process, even though the terms "machine architecture" and "machine language" are used. The implicit assumption is made that the HSI and DSI are the same with the exception of type B HLL machines, which differ from type A machines only in that the HSI is higher (above the DSI).

Thus, he discusses a category in which the hardware translates as well as interprets, but he seems to consider the "machine architecture" in this case to be the level from which interpretation takes place rather than the level from which translation takes place:

> "Note that the type B machine has the same semantic gap as a type A machine. Its only advantage over a type A machine is that the assembly process should be faster because it is implemented as a microprogram or in hardware." [20], p. 46

This would imply that he considers the DSI to define the semantic gap. However, in his discussion of hardware vs. software, he is clearly talking about something else:

> "Architects often use the following three criteria in determining whether a function should be implemented in the machine rather than in software: (1) the function should be small, (2) the function should be unlikely to change, and (3) system performance would suffer from a slower software implementation of the function." [20], p.46

These criteria are not related to the translation and interpretation issue and the second criterion clearly relates to a question of alterability. Thus, Myers shows that the HSI and the DSI (by our definitions) are separate things, even though he doesn't discuss them in this way.

Tanenbaum [34] separates the DSI from the HSI more clearly although he doesn't connect the two together. In Chapter 1, multilevel machines are introduced and the techniques of translation and interpretation are defined. He doesn't mention the DSI explicitly, but he does discuss translating to and interpreting from different levels (i.e., movement of the DSI). Then, software and hardware are discussed:

> "Any operation performed by software can also be built directly into the hardware and any instruction executed by the hardware can also be simulated in software. The decision to put certain functions in the hardware and others in the software is made on the basis of such factors as cost, speed, reliability, and frequency of expected changes." [34], p. 11

Thus, although the DSI and the HSI are not related to one another, they are clearly considered to be separate interfaces.

3. DSI/HSI CONFIGURATION EXAMPLES

Figure 1 shows an HSI versus DSI diagram showing where a number of computers fall within the two dimensional space. In this section we consider different approaches to computer design and discuss them in terms of how they affect the HSI and the DSI.

3.1. HIGH LEVEL LANGUAGE MACHINES

The principle behind the notion of high level language machines is to raise the HSI to the high level language level. The DSI is also generally raised as high as possible, usually slightly below the HSI. The SYMBOL machine was an early example of this [32], [33], [31]. In this case, the HSI is at the SYMBOL language level [7] since the hardware is able to accept SYMBOL language input, while the DSI is at a slightly lower level represented by the internal representation of a SYMBOL program. The hardware of the machine translates a SYMBOL program into this intermediate form (removes redundant blanks, changes keywords into bit strings, replaces symbolic addresses to pointers, etc.). After this translation has been completed for the entire program, execution begins.

Another example of a high level language machine is the Abacus machine that ran BASIC [3]. Like SYMBOL, this machine has the HSI at the high level language level and the DSI slightly below. Abacus did a hardware translation similar to that performed by SYMBOL before starting the execution of the program. A FORTRAN machine [1] also falls into this general category. The HSI is at the FORTRAN level, the DSI is slightly below. Finally, there is a machine proposed by Chu and Abrahms [5], [4]. Unlike the previous examples, this machine actually has the DSI at the high level language level. During the execution of the program, the hardware actually scans the source code and executes it. No translation is done previous to this and there exists no other specification of the program other than the source code.

3.2. DYNAMIC MICROPROGRAMMING

The basic idea of dynamic microprogramming is to lower the HSI to the single cycle level, while keeping the DSI at a typical level. The motivation behind dynamic microprogramming is to allow the DSI to be better matched to the problem being run. By lowering the HSI, the level in between the HSI and the DSI becomes dynamically alterable, thus, the DSI can be changed

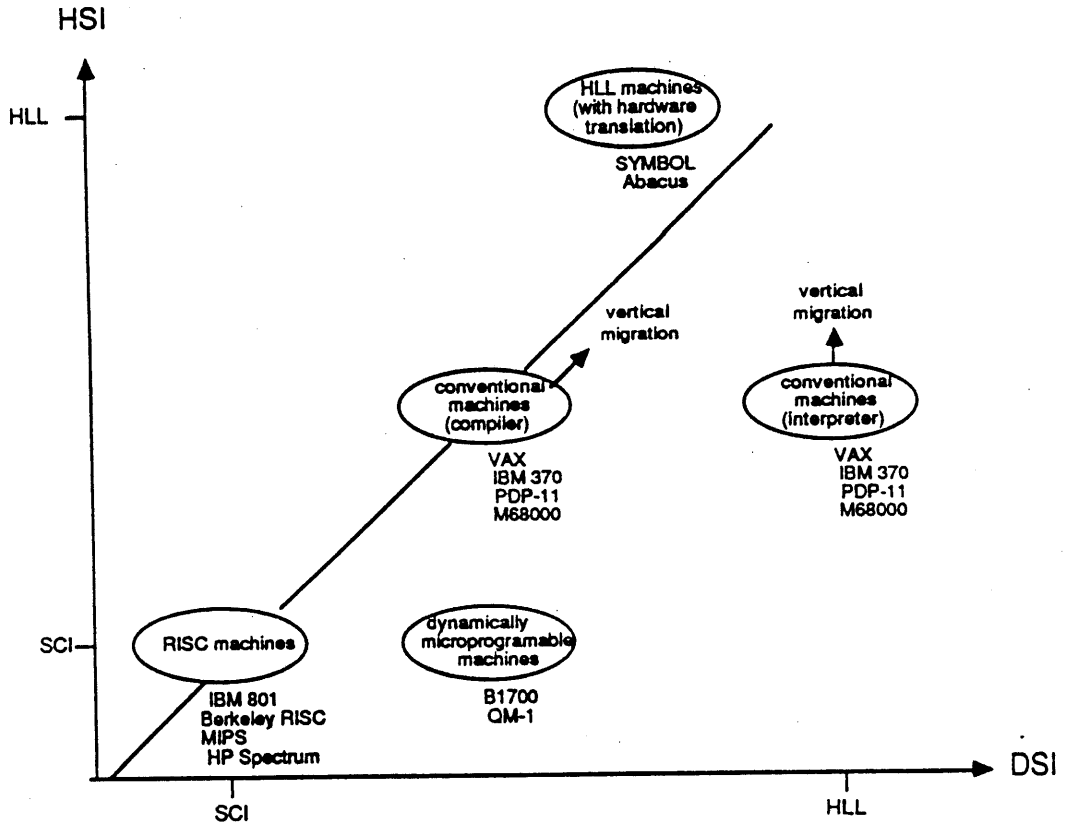

Figure 1, HSI / DSI Diagram

for different problems being run. Examples of dynamic microprogramming are the Burroughs B1700 series [21] and the QM-1[28].

Cook and Flynn describe a dynamically microprogrammable computer in [6], and Flynn, Neuhauser and McClure describe the EMMY system at Stanford in [10], which was similar. Also, the 'Interpreter' is described in [30], a system similar to the QM-1. Rauscher and Agrawala discuss the application of dynamically microprogrammable machines in [29]. Most of these papers and books, however, don't explicitly address the issue of DSI placement. The main point behind dynamic microprogramming is HSI placement, not DSI placement. It is generally assumed that the DSI should be where it typically has been. In other words, these machines are generally designed as interpreters, they are not set up to handle placement of the DSI at the HSI (compiling down to the microcode level).

3.3. VERTICAL MIGRATION

Another idea concerning the movement of the HSI and DSI is vertical migration. The principle here is to raise the HSI. In some cases the DSI is raised with the HSI, and in some cases it remains unchanged. The difference lies in whether the functions that are vertically migrated are then translated to, or are then used by an interpreter (see figure 1). In either case, the motivation behind this idea is to raise the HSI in order to allow more efficient execution of commonly executed functions. This is accomplished for two reasons. First, instruction bandwidth is reduced because more complex operations are encoded at the HSI level. Second, the implementor of the functions to be vertically migrated has a finer level of control than would be possible using constructs above the HSI. Typically, functions that were previously written in the macrocode of a conventional processor are rewritten in microcode, which then becomes part of the hardware.

Hassit and Lyon describe an application of vertical migration in [13] and [12]. This was a vertical migration of selected APL primitives. The primitives were microcoded on an IBM System/370 model 25. Weber describes an implementation of EULER on an IBM System/360 model 30 in [35]. Luque and Ripoll provide a summary and overview of vertical migration in [18]. Pihlgren describes the vertical migration of COBOL primitives in [25]. These are just a few examples of the many papers published in this area.

3.4. REDUCED INSTRUCTION SETS

The last idea about the HSI and DSI that will be discussed is the idea of "reduced instruction set" machines. There is no consensus on what defines a reduced instruction set machine. We will consider in this section only those machines that put the DSI and the HSI at the SCI, since we believe this is critical issue. Examples of these machines are the IBM 801 [26], the Berkeley RISC [23], [24], the Stanford MIPS [14], [15], and the HP Spectrum [2], [19]. Two other machines that have claimed to be reduced instruction set machines, the Ridge 32 [11], and the Pyramid 90X [27] may have lowered the HSI and the DSI somewhat compared with some machines, but they are both still above the SCI.

The unique feature of the reduced instruction set idea is to lower the DSI as well as the HSI. The dynamic microprogramming idea discussed above advocated the lowering of the HSI, but not the DSI. Many papers justifying the 'reduced instruction set' concept seem to ignore the fact that the DSI is lower as well as the HSI. For example, Radin, in [26], makes the following statement with regard to the IBM 801:

> " ... the benefits claimed [of microcode] are generally not due to the power of the instructions as much as to their residence in a high-speed control store. This amounts to a hardware architect attempting to guess which subroutines, or macros, are most frequently used and assigning high-speed memory to them. ... The 801 CPU gets its instructions from an 'instruction cache' which is managed by least-recently-used information. Thus, all frequently used functions are very likely to be found in this high-speed storage, ..." [26]

Birnbaum and Worley, in [2], p. 41, also make an equivalent remark about the HP Spectrum family. What they fail to make clear is that the DSI has been lowered as well as the HSI. Making the comparison between a control store and an instruction cache isn't as simple as they might have you believe because one has microinstructions below the DSI and the other has microinstructions above the DSI. These are two very different situations and a comparison is not simple. Besides, some computers cache the control store without lowering the DSI, for example the Burroughs B1726 [21]. The hit ratio of a control store cache depends only on the frequency of individual instructions. The hit ratio of an instruction cache, on the other hand, depends on instruction stream locality, compiler technology, and how dynamic the environment is.

4. CONCLUSIONS

The computer architect is faced with many tradeoffs. In order to appropriately evaluate these tradeoffs, many different tools are needed. In particular, it is necessary to capture the significant features of different designs while ignoring the insignificant ones. We have attempted to aid this process by clarifying the concept of the dynamic/static interface and comparing it to the hardware/software interface.

The interfaces themselves are not new, but we have combined them in a coherent manner and defined a two dimensional space that has each interface as an axis. This space, shown in figure 1, illustrates the usefulness of the HSI/DSI concept. Different approaches to computer design can be put into perspective with other approaches. Of course, this captures only some of the characteristics of a design, but we believe them to be significant ones.

This paper represents research in progress. Most of the ideas presented here are still developing. We welcome feedback and encourage discussion.

ACKNOWLEDGEMENT

We greatfully acknowledge our colleagues, students and faculty, within the Aquarius program for providing the stimulating environment at Berkeley within which we work. We also acknowledge that part of this work was sponsored by the Defense Advanced Research Projects Agency (DoD), Arpa Order No. 4871, monitored by Space and Naval Warfare Systems Command under Contract No. N00039-84-C-0089.

REFERENCES

[1] Theodore R. Bashkow, Azra Sasson, and Arnold Kronfeld, "System Design of a FORTRAN Machine," *IEEE Transactions on Electronic Computers*, Vol. EC-16, No. 4, August, 1967, pp. 485–499.

[2] Joel S. Birnbaum, and William S. Worley Jr., "Beyond RISC: High-Precision Architecture," *Proceedings of Spring COMPCON 86*, March 3–6, 1986, pp. 40–47.

[3] H. J. Bürkle, A. Frick, and Ch. Schlier, "High Level Language Oriented hardware and the Post-von Neumann Era," *Proceedings of The 5th Annual Symposium on Computer Architecture*, April 3–5, 1978, pp. 60–65.

[4] Yaohan Chu, "Architecture of a Hardware Data Interpreter," *IEEE Transactions on Computers*, Vol. C-28, No. 2, February, 1979, pp. 101–109.

[5] Yaohan Chu, and Marc Abrams, "Programming Languages and Direct-Execution Computer Architecture," *Computer*, Vol. 14, No. 7, July, 1981, pp. 22–32.

[6] Robert W. Cook, and Michael J. Flynn, "System Design of a Dynamic Microprocessor," *IEEE Transactions on Computers*, Vol. C-19, No. 3, March, 1970, pp. 213–222.

[7] David R. Ditzel, "Reflections on the High–Level Language Symbol Computer System," *Computer*, Vol. 14, No. 7, July, 1981, pp. 55–66.

[8] Michael J. Flynn, "Directions and Issues in Architecture and Language," *Computer*, Vol. 13, No. 10, October, 1980, pp. 5–22.

[9] Michael J. Flynn, "Towards Better Instruction Sets," *Proceedings of the 16th Annual Workshop on Microprogramming*, October 11–14, 1983, pp. 3–8.

[10] Michael J. Flynn, C. Neuhauser, and Robert M. McClure, "EMMY – An Emulation System for User Microprogramming," *AFIPS Conference Proceedings, The 1975 National Computer Conference*, Vol. 44, May 19–22, 1975, pp. 85–89.

[11] David Folger, and Ed Basart, "Computer Architectures – Designing for Speed," *Proceedings of Spring COMPCON 83*, February 28 – March 3, 1983, pp. 25–31.

[12] A. Hassitt, J. W. Lageschulte, and L. E. Lyon, "Implementation of a High Level Language Machine," *Communications of the ACM*, Vol. 16, No. 4, April, 1973, pp. 199–212.

[13] A. Hassitt, and L. E. Lyon, "An APL Emulator on System/370," *IBM System Journal*, No. 4, 1976, pp. 358–378.

[14] John Hennessy, Norman Jouppi, John Gill, Forest Baskett, Alex Strong, Thomas Gross, Chris Rowen, and Judson Leonard, "The MIPS Machine," *Proceedings of Spring COMPCON 82*, March 1982, pp. 2–7.

[15] John Hennessy, Norman Jouppi, Steven Przybylski, Christopher Rowen, Thomas Gross, Forest Baskett, and John Gill, "MIPS: A Microprocessor Architecture," *Proceedings of the 15th Annual Workshop on Microprogramming*, October 5–7, 1982, pp. 17–22.

[16] Lee W. Hoevel, "'Ideal' Directly Executed Languages: An Analytical Argument for Emulation," *IEEE Transactions on Computers*, Vol. C-23, No. 8, August, 1974, pp. 759–767.

[17] Harry Katzan Jr., *Computer Organization and the System/370*, Van Nostrand Reinhold Company, 1971.

[18] E. Luque, and A. Ripoll, "Microprogramming: A Tool for Vertical Migration," *Microprocessing and Microprogramming*, Vol. 8, 1981, pp. 219–228.

[19] J. Moussouris, L. Crudele, D. Freitas, C. Hansen, E. Hudson, R. March, S. Przybylski, T. Riordan, C. Rowen, and D. Van't Hof, "A CMOS RISC Processor With Integrated System Functions," *Proceedings of COMPCON 86*, March 3–6, 1986, pp. 126–131.

[20] Glenford J. Myers, *Advances in Computer Architecture, 2nd edition*, John Wiley and Sons, 1982.

[21] Elliott I. Organick, and James A. Hinds, *Interpreting Machines: Architecture and Programming of the B1700/B1800 Series*, Elsevier North–Holland, Inc., 1978.

[22] Yale N. Patt and John K. Ahlstrom, "Microcode and the Protection of Intellectual Effort," *Proceedings of the 18th Annual Workshop on Microprogramming*, Pacific Grove, CA, December 3–6, 1985, pp. 167–170.

[23] David A. Patterson, "A RISCy Approach to Computer Design," *Proceedings of Spring COMPCON 82*, March 1982, pp. 8–14.

[24] David A. Patterson, and Carlo H. Sequin, "A VLSI RISC," *Computer*, Vol. 15, No. 9, September, 1982, pp. 8–21.

[25] Göran Pihlgren, "Increasing Performance in a Cobol System by Adding Firmware," *EUROMICRO Journal*, Vol. 6, 1980, pp. 403–405.

[26] George Radin, "The 801 Minicomputer," *IBM Journal of Research and Development*, Vol. 27, No. 3, May 1983, pp. 237–246.

[27] Robert Ragan–Kelley, and Roy Clark, "Applying RISC Theory to a Large Computer," *Computer Design*, November, 1983, pp. 191–198.

[28] C. V. Rammamoorthy, and M. Tsuchiya, "A Study of User–Microprogrammable Computers," *AFIPS Conference Proceedings, 1970 Spring Joint Computer Conference*, Vol. 40, 1970, pp. 165–181.

[29] Tomlinson G. Rauscher, and Ashok K. Agrawala, "Dynamic Problem–Oriented Redefinition of Computer Architecture via Microprogramming," *IEEE Transactions on Computers*, Vol. C-27, No. 11, November, 1978, pp. 1006–1014.

[30] E. W. Reigel, U. Faber, and D. A. Fisher, "The Interpreter — A Microprogrammable Building Block System," *AFIPS Conference Proceedings, 1972 Spring Joint Computer Conference*, May 16–18, 1972, pp. 705–723.

[31] Rex Rice, "The Chief Architect's Reflections

on Symbol IIR," *Computer*, Vol. 14, No. 7, July, 1981, pp. 49–54.

[32] Rex Rice, and William R. Smith, "SYMBOL — A Major Departure From Classic Software Dominated von Neumann Computing Systems," *AFIPS Conference Proceedings, 1971 Spring Joint Computer Conference*, pp. 575–587.

[33] William R. Smith, Rex Rice, Gilman D. Chesley, Theodore A. Laliotis, Stephen F. Lundstrom, Myron A. Calhoun, Lawrence D. Gerould, and Thomas G. Cook, "SYMBOL — A Large Experimental System Exploring Major Hardware Replacement of Software," *AFIPS Conference Proceedings, 1971 Spring Joint Computer Conference*, pp. 601–616.

[34] Andrew S. Tanenbaum, *Structured Computer Organization, 2nd edition* Prentice-Hall, Inc., Englewood Cliffs, N. J., 1984.

[35] Helmut Weber, "A Microprogrammed Implementation of EULER on IBM System/360 Model 30," *Communications of the ACM*, Vol. 10, No. 9, September, 1967, pp. 549–558.

BIBLIOGRAPHY

[36] P. W. Agnew, and A. S. Kellerman, "Microprocessor Implementation of Mainframe Processors by Means of Architecture Partitioning," *IBM Journal of Research and Development*, Vol. 26, No. 4, July, 1982, pp. 401–412.

[37] B. Albert, A. Bode, and W. Händler, "A Case Study in Vertical Migration: The Implementation of a Dedicated Associative Instruction Set," *Microprocessing and Microprogramming*, Vol. 8, 1981, pp. 257–262.

[38] G. M. Amdahl, G. A. Blaauw, and F. P. Brooks Jr., "Architecture of the IBM System/360," *IBM Journal of Research and Development*, Vol. 8, No. 2, April, 1964, pp. 87–101.

[39] Takanobu Baba, Ken Ishikawa, and Kenzo Okuda, "A Two–Level Microprogrammed Multiprocessor Computer with Nonnumeric Functions," *IEEE Transactions on Computers*, Vol. C-31, No. 12, December, 1982, pp. 1142–1156.

[40] Helmut Berndt, "Software Support in Hardware," *Microprocessing and Microprogramming*, Vol. 13, 1984, pp. 1–9.

[41] Robert Bernhard, "More Hardware Means Less Software," *IEEE Spectrum*, Vol. 18, No. 12, December, 1981, pp. 30–37.

[42] Pradip Bose, and Edward S. Davidson, "Design of Instruction Set Architectures for Support of High–Level Languages," *Proceedings of The 11th International Symposium on Computer Architecture*, 1984, pp. 198–206.

[43] Charles W. Bridges, and Abd-Elfattah Mohamed Abd-alla, "Direct Execution of C-String Compiler Texts," *Proceedings of the 12th Annual Workshop on Microprogramming*, 1979, pp. 84–92.

[44] Richard P. Case, and Andris Padegs, "Architecture of the IBM System/370," *Communications of the ACM*, Vol. 21, No. 1, January, 1978, pp. 73–96.

[45] Gilman D. Chesley, and William R. Smith, "The Hardware–Implemented High–Level Machine Language for SYMBOL," *AFIPS Conference Proceedings, 1971 Spring Joint Computer Conference*, pp. 563–573.

[46] F. Chow, M. Himelstein, E. Killian, and L. Weber, "Engineering a RISC Compiler System," *Proceedings of Spring COMPCON 86*, March 3–6, 1986, pp. 132–137.

[47] Charles C. Church, "Computer Instruction Repertoire – Time for a Change," *AFIPS Conference Proceedings, 1970 Spring Joint Computer Conference*, 1970, pp. 343–349.

[48] F. C. Colon Osorio, and Y. N. Patt, "Tradeoffs in the Design of a System for High Level Language Interpretation," *Proceedings of the International Conference on Computer Design*, October, 1983.

[49] Robert P. Colwell, Charles Y. Hitchcock III, and E. Douglas Jensen, "Peering Through the RISC / CISC Fog: An Outline of Research," *Computer Architecture News*, Vol. 11, No. 1, March 1983.

[50] Robert P. Colwell, Charles Y. Hitchcock III, E. Douglas Jensen, H. M. Brinkley Sprunt, and Charles P. Kollar, "Computers, Complexity, and Controversy," *Computer*, Vol. 18, No. 9, September, 1985, pp. 8–19.

[51] D. S. Coutant, C. L. Hammond, and J. W. Kelly, " Compilers for the New Generation of Hewlett–Packard Computers," *Proceedings of Spring COMPCON 86*, March 3–6, 1986.

[52] David R. Ditzel, "Program Measurements on a High–Level Language Computer," *Computer*, Vol. 13, No. 8, August, 1980, pp. 62–71.

[53] Dennis A. Fairclough, "A Unique Microprocessor Instruction Set," *IEEE Micro*, Vol. 2, No. 2, May, 1982, pp. 8–18.

[54] Michael J. Flynn, "Trends and Problems in Computer Organizations," *Proceedings of the Information Processing Congress*, 1974, pp. 3–10.

[55] Michael J. Flynn, and Lee W. Hoevel, "Execution Architecture: The DELtran Experiment," *IEEE Transactions on Computers*, Vol. C-32, No. 2, February, 1983, pp. 156–175.

[56] Michael J. Flynn, and Lee W. Hoevel, "Measures of Ideal Execution Architectures," *IBM Journal of Research and Development*, Vol. 28, No. 4, July, 1984, pp. 356–369.

[57] Michael J. Flynn, John D. Johnson, and Scott P. Wakefield, "On Instruction Sets and Their Formats," *IEEE Transactions on Computers*, Vol. C-34, No. 3, March, 1985, pp. 242–254.

[58] Barbara Grossmann, Eva Kwee, and Axel Lehmann, "Practical Experiences with Vertical Migration," *Microprocessing and Microprogramming*, Vol. 12, 1983, pp. 185–192.

[59] Paul M. Hansen, Mark A. Linton, Robert N. Mayo, Marguerite Murphy, and David A. Patterson, "A Performance Evaluation of the Intel iAPX 432," *Computer Architecture News*, Vol. 10, No. 4, June, 1982.

[60] J. L. Heath, "Re–Evaluation of the RISC I," *Computer Architecture News*, Vol. 12, No. 1, March, 1984, pp. 3–10.

[61] Charles Y. Hitchcock, and H. M. Brinkley Sprunt III, "Analyzing Multiple Register Sets," *Proceedings of The 12th Annual International Symposium on Computer Architecture*, June 17–19, 1985, pp. 55–63.

[62] Norbert Hojka, "Increasing Performance by Microcoding," *Microprocessing and Microprogramming*, Vol. 8, 1981, pp. 229–236.

[63] Bernhard Holtkamp, "UNIX Requirements for Arhcitectural Support," *Microprocessing and Microprogramming*, Vol. 15, 1985, pp. 129–140.

[64] Martin E. Hopkins, "A Perpective on Microcode," *Proceedings of Spring COMPCON 83*, February 28 – March 3, 1983, pp. 108–110.

[65] William C. Hopkins, "HLLDA Defies RISC: Thoughts on RISC's, CISC's and HLLDA's," *Proceedings of the 16th Annual Workshop on Microprogramming*, October 11–14, 1983, pp. 70–74.

[66] Miquel Huguet, and Tomas Lang, "A Reduced Register File for RISC Architectures," *Computer Architecture News*, Vol. 13, No. 4, September, 1985, pp. 22–31.

[67] Krishna Kavi, Boumediene Belkhouche, Evelyn Bullard, Lois Delcambre and Stephen Nemecek, "HLL Architectures: Pitfalls and Predilections," *Proceedings of the 9th Annual Symposium on Computer Architecture*, April 26–29, 1982, pp. 18–23.

[68] E. Korthauer, and L. Richter, "Are RISCs Subsets of CICSs? A Discussion of Reduced versus Complex Instruction Sets," *Microprocessing and Microprogramming*, Vol. 14, 1984, pp. 1–8.

[69] Harold W. Lawson Jr., "Programming–Language–Oriented Instruction Streams," *IEEE Transactions on Computers*, Vol. C-17, No. 5, May, 1968, pp. 476–485.

[70] Amund Lunde, "Empirical Evaluation of Some Features of Instruction Set Processors Architectures," *Communications of the ACM*, Vol. 20, No. 3, March 1977, pp 143–153.

[71] Emilio Luque, Ana Ripoll, and Jose J. Ruz, "Dynamic Microprogramming in Computer Architecture Redefinition," *EUROMICRO Journal*, Vol. 6, 1980, pp. 98–103.

[72] W. Mackrodt, "Considerations on Language Interpretation for Microprocessor Systems," *Microprocessing and Microprogramming*, Vol. 7, 1981, pp. 110–118.

[73] Richard L. Mandell, "Hardware/Software Trade–Offs — Reasons and Directions," *AFIPS Conference Proceedings, 1972 Fall Joint Computer Conference*, Vol. 41, 1972, pp. 453–459.

[74] Richard J. Markowitz, "Software Impact on Microcomputer Architecture, A Case Study," *Proceedings of the 8th Annual Symposium on Computer Architecture*, 1981, pp. 40–48.

[75] Motorola, Inc., *MC68020 32-Bit Microprocessor User's Manual*, Prentice–Hall, Inc., 1984.

[76] Richard L. Norton, and Jacob A. Abraham "Adaptive Interpretation as a Means of Exploiting Complex Instruction Sets," *Proceedings of The 10th Annual International Symposium on Computer Architecture*, 1983, pp. 277–282.

[77] Monika Obrebska, "Efficiency and Performance Comparison of Different Degisn Methodologies for Control Parts of Micrprocessors," *Microprocessing and Microprogramming*, Vol. 10, 1982, pp. 163–178.

[78] David A. Patterson, Phil Garrison, Mark Hill, Dimitris Lioupis, Chris Nyberg, Tim Sippel, and Korbin Van Dyke, "Architecture of a VLSI Instruction Cache for a RISC," *Proceedings of The 10th Annual International Symposium on Computer Architecture*, 1983, pp. 108–116.

[79] David Patterson, and John Hennessy, "Response to 'Computers, Complexity, and Controversy'," *Computer*, Vol. 18, No. 11, November, 1985, pp. 142–143.

[80] David A. Patterson, and Richard S. Piepho, "Assesing RISCs in High–Level Language Support," *IEEE Micro*, Vol. 2, No. 4, November, 1982, pp. 9–19.

[81] David A. Patterson, and Carlo H. Sequin, "Design Considerations for Single–Chip Computers of the Future," *IEEE Transactions on Computers*, Vol. C-29, No. 2, February, 1980, pp. 108–116.

[82] *PDP-11 Architecture Handbook*, Digital Equipment Corporation, 1983.

[83] Richard S. Piepho, "Comparative Evaluation of the RISC I Architecture via the Computer Family Architecture Bechmarks," *University of California at Berkeley, EECS Master's Report*, August 27, 1981.

[84] Peter U. Schulthess, "A Reduced High–Level–Language Instruction Set," *IEEE Micro*, Vol. 4, No. 3, June, 1984, pp. 55–67.

[85] Robert J. Sheraga, and John L. Gieser, "Experiments in Automatic Microcode Generation," *IEEE Transactions on Computers*, Vol. C-32, No. 6, June, 1983, pp. 557–569.

[86] John A. Stankovic, "The Types and Interactions of Vertical Migrations of Functions in a Multilevel Interpretive System," *IEEE Transactions on Computers*, Vol. C-30, No. 7, July, 1981, pp. 505–513.

[87] John Stockenberg, and Andries van Dam, "Vertical Migration for Performance Enhancement in Layered Hardware/Firmware/Software Systems," *Computer*, Vol. 11, No. 5, May, 1978, pp. 35–50.

[88] Harold S. Stone, *Introduction to Computer Architecture, 2nd Ed.*, Science Research Associates, Inc., 1980.

[89] Yuval Tamir, "Simulation and Performance Evaluation of the RISC Architecture," *Electronics Research Laboratory, University of California, Berkeley*, Memorandum No. UCB/ERL M81/17, April 6, 1981.

[90] Yuval Tamir, and Carlo H. Sequin, "Strategies for Managing the Register File in RISC," *IEEE Transactions on Computers*, Vol. C-32, No. 11, November, 1983, pp. 977–989.

[91] Andrew S. Tanenbaum, "Implications of Structured Programming for Machine Architecture," *Communications of the ACM*, Vol. 21, No. 3, March, 1978, pp. 237–246.

[92] Richard T. Thomas, "Organization for Execution of User Microprograms from Main Memory: Synthesis and Analysis," *IEEE Transactions on Computers*, Vol. C-23, No. 8, August, 1974, pp. 783–790.

[93] Nick Tredennick, "The 'Cultures' of Microprogramming," *Proceedings of The 15th Annual Microprogramming Workshop*, 1982, pp. 79–83.

[94] Philip C. Treleaven, "VLSI Processor Architectures," *Computer*, Vol. 15, No. 6, June, 1982, pp. 33–45.

[95] Allen B. Tucker, and Michael J. Flynn, "Dynamic Microprogramming: Processor Organization and Programming," *Communications of the ACM*, Vol. 14, No. 4, April, 1971, pp. 240–250.

[96] Maurice V. Wilkes, "The Processor Instruction Set," *Proceedings of the 15th Annual Workshop on Microprogramming*, October 5–7, 1982, pp. 3–5.

[97] W. T. Wilner, "Design of the Burroughs B1700," *AFIPS Conference Proceedings, 1982 National Computer Conference*, December 5–7, 1982, pp. 489–497.

[98] William A. Wulf, "Compilers and Computer Architecture," *Computer*, Vol. 14, No. 7, July, 1981, pp. 41–47.

[99] *VAX Architecture Handbook*, Digital Equipment Corporation, 1981.