EVALUATION OF ALTERNATIVE COMPUTER ARCHITECTURES

M.R. Barbacci[1], W.E. Burr[2],
S.H. Fuller[1], and D.P. Siewiorek[1] (Eds.)

Department of Computer Science
Carnegie-Mellon University
Pittsburgh Pa. 15213

February 12, 1977

The Computer Family Architecture project was organized and supervised by the Army Electronics Command, Fort Monmouth, N.J. and the Naval Research Laboratory, Washington D.C. The work described here was supported in part by the Defense Advanced Research Projects Agency (ARPA) under contract F44620-73-C-0074, in part by the National Science Foundation under grant GJ 32758X, and in part by the Army Research Office under grant DAAG29-76-G-0299.

1 Also with the Naval Research Laboratory, Washington, D.C.

2 U.S. Army Electronics Command, Ft. Monmouth, N.J.

Abstract

The Computer Family Architecture (CFA) Selection Committee was organized to select a proven, well-known computer architecture, in addition to several widely used military computer architectures, as the basis of the future series of Military Computer Family (MCF) computers. The set of four papers that make up this report provide an overview of the work of the CFA Committee and a detailed discussion of the technical methods used to quantitatively evaluate the alternative computer architectures under consideration.

As the first paper describes, support software availability, life cycle costs, and architecture licensing, in addition to architectural efficiency, were considered in the final evaluation process. As a result of this process, the CFA Committee ranked the three architecture finalists in the following order: the DEC PDP-11, the IBM System/370, and the Interdata 8/32. The MCF project is now working on the specification of a new standard architecture for military applications based on the PDP-11. In addition, the MCF project is working on more clearly specifying the most widely used existing military computer architectures to enable future re-implementations of these architectures in new technologies.

Papers in this Technical Report

1. Burr, W.E., A.H. Coleman, and W.R. Smith: Summary of the Final Report of the Computer Family Architecture Selection Committee

2*. Fuller, S.H., W.E. Burr, and S.H. Stone: Initial Selection and Screening of the CFA Candidate Computer Architectures

3*. Fuller, S.H., W.E. Burr, P. Shaman, and D. Lamb: Evaluation of Computer Architectures via Test Programs

4*. Barbacci, M.R., D.P. Siewiorek, R. Gordon, R. Howbrigg, and S. Zuckerman: <u>Architecture Research Facility: ISP Descriptions, Simulation, and Data Collection</u>

* These papers will appear in the 1977 National Computer Conference, Dallas, Texas, June 13-16, 1977.

SUMMARY OF THE FINAL REPORT OF THE ARMY/NAVY COMPUTER FAMILY ARCHITECTURE SELECTION COMMITTEE

William E. Burr
U.S. Army Electronics Command
Fort Monmouth, N.J.

Aaron H. Coleman
U.S. Army Electronics Command
Fort Monmouth, N.J.

and

William R. Smith
Naval Research Laboratory
Washington, D.C.

Summary of the Final Report

TABLE OF CONTENTS

Summary of the Final Report

Abstract

An Army/Navy Computer Family Architecture (CFA) Selection Committee, comprising 10 Army and 17 Navy organizations was organized by the Naval Research Laboratory and the Army Electronics Command in 1975 to select a proven, well-known computer architecture to be the basis of a Military Computer Family (MCF). The Selection Committee met five times in the period between October, 1975, and August, 1976, and evaluated nine computer architecture candidates in accordance with criteria established by the Committee. The Committee applied a preliminary screening process to select three candidates (IBM S/370, DEC PDP-11, and Interdata 8/32) for more intensive evaluation. This final evaluation process considered experimentally determined architectural efficiency, support software availability, life cycle cost, and architecture licensing. As a result of this process, the Committee ranked the three architecture finalists in the following order:

1. PDP-11
2. S/370
3. 8/32

1. Introduction

This report describes the work performed by an Army/Navy Committee, representing 10 Army and 17 Navy organizations, to select a Computer Family Architecture (CFA) for use with a proposed software compatible family of military computers and associated systems/support software. This family is known as the Military Computer Family (MCF).

This report summarizes the contents of a full report on the work of the CFA Selection Committee (i.e., "Final Report of the CFA Selection Committee"). References to this full report will be made herein, in accordance with the table of contents shown in the Appendix.

2. Background

The Department of Defense is spending over six billion dollars yearly for ADP systems. A large portion of this goes for acquisition of militarized computers and associated software that are used in tactical and strategic areas. Traditionally, these computers have been specified by the individual organizations (military project offices or commercial contractors) responsible for the development of each system. More often than not, computer selections are based upon local schedule, funding, or profit considerations, rather than the impact that the selection would have on long range hardware/software logistics costs. The result has been that the large number of types of computers used in Army and Navy systems are causing serious problems in the development and maintenance of software for those systems.

Military computers are usually procured as integral components of larger systems (e.g., radars, missile systems); the hardware issues have historically been given more attention than the logistics of the software, and in consequence, military computers normally have only the most primitive sort of support software. The development cycles for weapons systems are generally long enough (5 to 10 years) that the military computers in these systems are often obsolete before they are ever delivered to the Field Army or the Fleet. Past computer standardization efforts in the military have concentrated on hardware packaging or obscure architectures of such small market that there has been no incentive for the computer industry at large to invest in developing software and hardware compatible with these computers. The end result of these conditions is that the military pays over and over for development of computer systems that frequently fall far short of performance expectations.

This can be contrasted with the situation in the commercial OEM (original equipment manufacturer) marketplace. Here computers are produced for the much larger commercial market by the thousands or even the tens of thousands. A number of manufacturers such as DEC, Data General, and Interdata have software compatible product lines, covering a wide range of processors of varying capabilities. Due to fierce competitive market pressures, system deficiencies are corrected, or the systems disappear. New products are developed much more quickly, and full advantage is taken of the advances in semiconductor device technology. Finally, due to the much larger user bases of commercial computers, and the competitive pressures of the marketplace, the support software bases of successful commercial computers are usually far superior to their military equivalents and are frequently improved or augmented by organizations seeking a share of this market.

A solution to many of the software problems with contemporary military computers would be to produce a family of software-compatible militarized computers. Moreover, if such a family were based upon a proven, commercial instruction-set architecture, then it would be possible to capture a good mature support software base, and to be certain that any architectural shortcomings were known and recognized. As the commercial system evolved, and the architecture was extended to meet the competition, the military computer family could also take advantage of these same extensions. Adhering to an established family in this way would avoid the architectural mavericks that limited-production military computers are prone to be.

3. The CFA/MCF Project

Since early 1975, the Center for Tactical Computer Sciences (CENTACS) of the U. S. Army Electronics Command and the Naval Air Systems Command (NAVAIR) have been supporting a cooperative Army/Navy effort to develop such a family of military computers, based upon a common instruction-set architecture.

The fundamental premise of the MCF project is that software compatibility should be achieved by the adoption of an existing, proven computer architecture for the MCF, thereby minimizing the risks inherent in the design of a new computer architecture and permitting the "capture" of an existing and evolving software base. In this context, computer architecture is distinguished from implementation considerations, and is defined as the structure of the computer which a machine level programmer needs to know in order to write all programs which will run correctly on the computer. For example, the architecture of the IBM S/370 is defined in the IBM System/370 Principles of Operations Manual. There are many implementations of the architecture (370-158, 370-168, etc.), but only one architecture, and every implementation will execute the same software. Another premise upon which the Army/Navy cooperative effort is based is the goal of software transportability from prior generation military computers to the MCF, most probably via emulation. In other words, the Army and Navy cannot abandon its investment in existing software. There is a strong analogy here with IBM's continued support of such machines as the 1401 and the 7090 via emulation, when the 360 family was introduced.

The first task of the MCF project was the selection of the CFA. CENTACS and the Naval Research Laboratory cooperated to lead that effort, and the following sections of this report describe how that selection was made.

The second task of the project is to develop a System Implementation Plan, which in a commercial organization would probably be called a product plan, to define the form, fit, and function characteristics of the MCF and the individual family members. The instruction-set architecture of the processors, not the detailed logic design will be specified, so that various military equipment manufacturers (in general, <u>not</u> the manufacturer of the commercial version of the CFA) will be able to independently develop MCF members to meet the form, fit, and function requirements of the MCF, and to run the CFA instruction set. This approach will permit multiple sources for the various family members, and will allow manufacturers to take maximum advantage of rapidly developing semiconductor technology. The goal is a line of plug-compatible modules that can be interconnected as computer systems in a variety of configurations, to meet a wide range of performance/ application requirements.

A similar Support Software Implementation Plan contract is planned for FY 1978. This plan will attempt to take maximum advantage of the existing support software base for the selected CFA.

4. The CFA Selection Committee

The mechanism for selecting the CFA was a joint Army/Navy Selection Committee. In order to achieve a wide representation of military computer requirements in this effort, letters were sent to Army and Navy Laboratories, System Centers, and Project Managers, inviting them to nominate "candidate" architectures, and to participate in the CFA selection process as members of the CFA Selection Committee. Ten Army and 17 Navy organizations assigned representatives to the Selection Committee, which was active between October 1975, and August 1976. The members and officers of the Selection Committee are given in Volume I of the Final Report.

Of the several procedural rules adopted by the Committee, the most important was the requirement for a 2/3 vote of the members present to carry a committee motion.

5. Candidate Architectures

The basic mechanism for deciding which architectures should be considered by the committee was to ask Army and Navy organizations to nominate candidate architectures. These nominations were augmented by the Committee in its early meetings. The architectures which were considered by the Committee are:

Burroughs B-6700
DEC PDP-11
IBM S/370
Interdata 8/32
Litton An/GYK-12
NOVA/ROLM 1662
Systems Engineering Laboratories SEL 32
Univac AN/UYK-7
Univac AN/UYK-20

On the list of candidates the S/370 and the B6700 are large scale commercial data processing type architectures. The PDP-11, SEL-32, 8/32, and the NOVA are classical OEM type minicomputers, and the AN/GYK-12, AN/UYK-7, and the AN/UYK-20 are three of the most widely used military computers.

Although the above list of architectures is not all inclusive, most of the Army and Navy organizations who nominated candidates went through their own internal screening process, considering a much wider selection of architectures prior to making their nominations. As a result, the nine architectures considered by the Committee represent the best candidates for a family of computers for military applications, according to the consensus of over two dozen Army and Navy organizations.

6. Selection Procedure

It was apparent to the Committee after much discussion, that there were certain key, critical characteristics that should be well satisfied by the selected CFA. Further, it became apparent that it made sense to perform an initial screening and ranking of the candidates, based on these characteristics, so that the obviously least acceptable candidates could be discarded and those with the most potential could be retained and investigated much more thoroughly. An initial screening process was therefore devised to select several "best final candidates" for more detailed evaluation.

After the initial screening process was completed, the three final candidates were subjected to a test program experiment to measure the efficiency of the architectures. The support software bases of the three architectures were studied, and life-cycle cost models were constructed to determine if one of the three architectures had a decisive economic advantage. Finally, the manufacturers were contacted to determine the conditions under which they would be willing to license their architectures for production by military vendors. This process is illustrated in Figure 1, and is described in more detail below.

6.1. Initial Screening

The Selection Committee decided to select the final candidate architectures from the initial list by means of two kinds of criteria. The first kind of criteria, which served as pass/fail tests of architectural adequacy, were called "absolute criteria". The committee planned to eliminate all architectures which did not completely satisfy these criteria. Absolute criteria included such requirements as a satisfactory protection mechanism, and a virtual to physical address translation mechanism. The second kind of criteria were caled "quantitative criteria". The quantitative criteria were intended to provide a relative ranking of the architectures in terms of characteristics which the committee believed were important measures of a computer architecture. Quantitative criteria included such characteristics as the size of the physical address space, the size of the virtual address space, the number of bits which had to be moved to save that state of the machine under various circumstances, and the size of the installed user base. A listing and very brief description of the absolute and quantitative criteria are shown in Table 1. The reader should see Volume II of the CFA Committee Final Report for a detailed discussion of these criteria. Each quantitative criterion was assigned a weighing factor by each committee member organization. An average weighing factor was computed for the entire committee for each criterion. The quantitative criteria scores for each candidate were normalized, weighted, and summed to give a composite figure of merit for each architecture.

Subcommittees were created to evaluate each architecture, in terms of the absolute and quantitative criteria. A meeting of the full committee was then devoted principaly to verifying the consistency and correctness of the evaluations of the candidate architectures. In addition, the results of this evaluation were audited by a consultant to ensure the consistency and correctness of the evaluation.

A principal difficulty in making the evaluations was the imprecision of most of the reference manuals of the candidate architectures, requiring frequent communication with the manufacturers in some cases. Certain of the manuals, as typified by the IBM S/370 Principals of Operation Manual, appeared to be complete and precise definitions of an architecture. Others left essential architectural details ambiguously defined or not defined at all.

The results of the absolute and quantitative criteria evaluations are summarized in Table 2. The PDP-11 and the IBM S/370 were the only two architectures which clearly passed all the absolute criteria, and they also were among the top three in the quantitative criteria evaluation. The Interdata 8/32 was also selected as a finalist on the basis of its very strong showing on the quantitative criteria, despite a nagging technical uncertainty concerning the state of the machine after interrupts, which the committee was never able to resolve to its own satisfaction.

The reader is cautioned that the application of these criteria requires a great deal of interpretation. The Selection Committee went to some considerable effort to arrive at comparable interpretations for each architecture. It may not be at all obvious from the simple definitions presented here, how the actual values used by the committee were calculated. This is documented in detail in Volume II of the CFA Committee Final Report, and the interested reader should refer to Volume II.

6.2. Final Candidates Evaluation

Architecture Efficiency Evaluation

A Test Program Subcommittee was appointed at the first Selection Committee meeting. This subcommittee proposed a set of 23 potential test programs, which were believed to be representative of the operations performed in military data processing applications. The Committee ranked these programs by their relative importance, and the top 12 programs were selected as the basis of the Test Program Experiment. These 12 programs are listed and briefly described in Table 3.

Each of the 12 test programs was a relatively small kernel-type program, most were subroutines, and most were defined as "structured" programs in a Program Definition Language (PDL). Programmers were then asked to "hand compile" the programs into the assembly languages of the respective machines from their PDL descriptions. This procedure was followed to minimize the effects of programmer variations. No large scale programs from "real" military systems were coded, because of the excessive expense involved in coding and testing a statistically significant set of such programs. High level language programs were not tested, because there is no practical was to separate the effects of compiler efficiency from the effects of architecture efficiency which the experiment was intended to measure.

Slightly over one hundred test program samples were coded by 16 programmers at participating organizations. The experiment was designed using analysis of variance techniques to give the best possible estimates of the relative efficiency of the three architectures.

Three measures were defined to gauge the efficiency of the architectures, independently of hardware implementation features such as cycle time. These measures were:

S The static storage requirement for the program in bits.

M The number of bits of program and data which were transferred between the processor and main memory during execution of a program. The M Measure is intended to be an index of the memory bandwidth requirements of an architecture.

R The number of bits of program and data which were transferred among the internal processor registers during execution of a program. The R Measure is intended to be an index of the processor bandwidth requirements of an architecture.

The S, M and R measures are indicators of the relative amounts of hardware capability that are necessary when implementing an architecture to do a certain job.

That is, larger S measure means that correspondingly more memory will be required to handle a given set of applications programs. Clearly, the architecture that can execute the programs with the smallest S is desirable. Similarly, M and R are indicators of the relative hardware speed/bandwidth requirements for memory and processor implementations.

The S, M and R raw data were gathered with the help of a special ISP language compiler and simulator system. The three architectures were described in ISP (Instruction Set Processor), a formal language for describing computers at the instruction/register level. These ISP descriptions were then compiled and run on the ISP simulator which was designed to automatically gather statistics of register and memory activity during execution of the test programs on the simulated candidate architectures. See Volume IV of the Committee Report for a detailed treatment of the ISP System and its use in the CFA effort.

The final results reflect the performance of each candidate architecture for each measure. Those results are shown in Table 4. This experiment is described more fully in Volume III of the final Committee Report.

The results are normalized so that unity indicated average performance; the lower the score on any of the measures, the better the architecture handled the set of test programs. In other words, the results indicate that the S/370 needs 21 percent more memory than the average to store the test programs, the 8/32 needs only 83 percent as much memory as average, and the PDP-11 is nearly average in its use of memory. The differences between the S/370 results and the average of the results of the other two architectures were statistically significant at the 95 percent confidence level, but the differences between the 8/32 and the PDP-11 results were not statistically significant at this confidence level. The differences between the 8/32 and the S/370 results were also statistically significant for the S and M measures at the 95 percent confidence level.

<u>Support Software Evaluation</u>

A support Software Evaluation Subcommittee was appointed to study the support software bases of the three final candidate architectures. This subcommittee began by defining an extensive menu of support software tools, which might be useful in military systems development. Committee member organizations were then asked to rate each tool by its utility in developing software for military weapon systems. The 28 most important support software tools were selected from this rating. The CFA candidate manufacturers and other commercial sources were investigated as to the availability of these 28 software tools for each architecture. Table 5 lists the basic tool types on the required support software menu.

The cost to develop each item of support software was estimated. The total cost to develop the selected support software items was estimated to be approximately 41 million dollars. The estimated value of the support software bases for each of the final candidate architectures is summarized in Table 6 below; also shown is the estimated cost to eliminate deficiencies as compaed to the desired support software base:

See Volume V of the Committee Report for a detailed treatment of the support software evaluation.

<u>Life Cycle Cost Evaluations</u>

A Final Selection Methodology Subcommittee was formed at the third Selection Committee meeting to investigate and pursue a methodology for combining the results of the committee's evaluations into a single evaluation criterion which would be realistic and meaningful to DoD management. This subcommittee proposed a method of converting the architecture and software evaluation results to life cycle costs so that a final selection could be aided by data based on the comparative economics of using each of the candidate architectures in military computer systems.

Two separate computer life cycle requirements models were used for the cost analyses. Both used the data gathered in the Architecture Efficiency Evaluation and the Support Software Base Evaluation described previously to convert the modeled requirements into dollar costs.

The first model is a "top-down" model which represents total life cycle requirements for DoD computers in the 1978-1990 time period, using each of the three final candidate architectures for the MCF. It was based upon extrapolating trends in DoD wide expenditures and requirements for military computer hardware and software.

Figure 2 summarizes results of computing CFA life cycle costs summed over the years 1978 to 1990 for the three candidates, for certain conditions. To simplify comparisons, the total assumed costs (approximately $1 billion) are normalized with respect to the IBM S/370.

The results are shown for specific values of two of the model input parameters. The first is an expenditure rate ($2M/year) for completing development of the support software base of each candidate. The second is a range of values (x-axis) of the total cost ratio of software-to- hardware for military tactical computer systems. The results are plotted as a function of the software-hardware cost ratio because it is one of the most important parameters in the cost evaluations. Available data gives this ratio as about 2.5 to 3.0 for generalized ADP systems but less than that for tactical, embedded computers where many copies of a single hardware and software design are deployed. How much less is not clear from available data. In the lower range of software/hardware ratios the Interdata has the lowest cost, in the upper range the S/370 is lowest, and the PDP-11 is lowest in the middle range and neither best nor worst elsewhere.

The second model is a "bottom-up" life cycle requirements model, which is based upon data gathered on 15 existing or developmental Army tactical data systems. This model represented the life cycle requirements for these 15 systems, using each of the three final candidate architectures. The cost to develop all of these systems in 1976 and then in 1985 was estimated. The results of this analysis are shown in Table 7 below. This table indicates that:

a. The average total life cycle cost for all 15 systems is estimated at $1.91B in 1976 and $250M in 1985. The average software: hardware cost ratio of these systems is 1:11 in 1976 and 1:2.3 in 1985.

b. In 1976, the number of systems in which the PDP-11 architecture provides the lowest cycle cost is the largest (11). The PDP-11 architecture provides the lowest total life cycle cost by a small margin (3.7%) over the 8/32 architecture and by a larger margin (20.0%) over the S/370 architecture.

c. In 1985, the number of systems in which the PDP-11 architecture provides the lowest cost increases to 14. The PDP-11 architecture continues to provide the lowest total life cycle cost for all 15 systems by margins of 8.8% and 17.6% over the 8/32 and S/370 architectures.

Assumptions applicable to the results shown in Table 7 are (1) hardware cost reduction of a factor of 10 from 1976 to 1985, (2) hardware life cycle cost of twice the acquisition cost, and (3) software life cycle cost of 5.5 times the acquisition cost.

The results shown in Table 7 are not significantly sensitive to changes in applications software cost or in the annual support software investment for the selected CFA.

A limited sensitivity analysis was performed with both models. If lower estimates are made for software development costs (relative to hardware costs), and/or if faster development of the support software base is projected (so that all three architectures rapidly acquire a complete support software base), then the Interdata 8/32 eventually becomes the least expensive architecture, because of its efficient architecture as indicated by the test program results. If very high software development cost estimates are made, and/or very slow support software development is projected, then the S/370 becomes the least expensive architecture because of its advantage in support software. Figure 2 illustrates this behavior. In the intermediate ranges of software cost estimates, where top-down and bottom-up results were in the best agreement, the PDP-11 appears to have a slight cost advantage. However, compared to the expected errors in the results due to the uncertainties in the input data and assumptions, the life-cycle cost differences between the two models and among the three candidate architectures are small. The software/hardware ratio which is one of the most important factors in both models is one of the hardest to pin down with supporting data, and the results of both models can be made to change by using values from different sources for the same input parameters. The strongest conclusion to be derived is that the results agree and that, in terms of life cycle cost, all three candidates would provide comparable choices for the CFA. See Volume VI of the Final Report for details of the life-cycle cost evaluations.

Licensing

Meetings were held with IBM, DEC, and Interdata to discuss the terms and conditions under which they would grant a non-exclusive license to the Government to use their architecture for militarized processors. All three manufacturers were cooperative and proposed terms for such an agreement. Although the proposed licensing agreements were a significant factor in the final selection process, the details cannot be given here, due to the confidential nature of the discussions. Volume VII of the Final Report, which is restricted to internal Government use, contains the details of the licensing proposals.

6.3. Final Selection/Recommendations

The Selection Committee held its fifth and final meeting on 24 to 26 August 1976 at the Naval Underwater Systems Center, Newport, R. I., for the purpose of selecting the recommended aachitecture for the MCF. At this meeting, the results of the evaluations discussed in the preceding sections of this article were considered by the committee and discussed at length. Based upon that data, and upon other concerns specifically considered by the committee during its discussion of the final selection, the respective strengths and weaknesses of each architecture can be summarized as follows:

A. INTERDATA 8/32. The 8/32 was the highest rated architecture on the Quantitative Criteria, and the Test Program Results. The 8/32 has a good interrupt structure for real-time processing. On the other hand, the software base is relatively weak, which consequently compromised its performance in the life cycle cost evaluations. There was a nagging question about how well the state of the machine was preserved after interrupts.

B. IBM S/370. The strongest virtue of the S/370 is its large support software base. The S/370 performed well on the life-cycle cost analyses under assumptions of maximum relative cost of software development. The S/370 is the only architecture demonstrated as an easily virtualized computer in a standard product line. On the other hand, its interrupt structure was considered cumbersome for real time control applications. The test program results indicate that the architecture is significantly less efficient than the 8/32 and the PDP-11. There was also concern that small subset versions might not prove cost-effective for low-end applications, and that there was insufficient experience with the S/370 in OEM type applications.

C. PDP-11. The PDP-11 enjoys a good support software base, performed relatively well on the Test Programs, and has a good interrupt structure for real-time control applications. It enjoys a slight advantage on the cost models for a range of reasonable assumptions. Small scale (microprocessor) implementations are practical and have been built. On the negative side, the 16 bit virtual address space is a limitation and it may be expensive to add a virtual machine capability to the architecture.

The committee made four final recommendations:

A. The DEC PDP-11 was determined by a vote of 14 to 4 to be the most advantageous architecture for the MCF, the IBM S/370 was ranked second, and the Interdata 8/32 was ranked third.

B. The committee unanimously agreed that a single instruction-set architecture should be selected for the MCF, that the selection of only one architecture is more important than which one of the candidates is selected, and that any one of the three final candidate architectures could provide a satisfactory basis for the MCF.

C. The committee agreed that an effort should be made to relieve the limitations of the selected architecture. In the case of the PDP-11 the major limitation is the small (16 bit) virtual address space.

D. A single organizational structure must be established to control the architecture, or major incompatibilities between different implementations will surely result.

See Volume VIII of the Final Report for details of the CFA final selection/recommendation process.

7. Conclusions

It is sometimes asserted that military systems have unique requirements which preclude the use of a general purpose commercial instruction set. Developers of computer based weapons systems often assert that they alone have such severe "real-time" constraints that they compel the use of a particular processor. It is worth noting that the Selection Committee compared three of the most widely used military architectures with six of the most widely used commercial architectures and found that the military architectures were deficient compared to the commercial architectures in terms of those architectural characteristics believed to be most important in tactical military applications. It is worth noting also that none of the military architectures had any unique features which proved advantageous, while all three were found to have architectural shortcomings. Moreover, the support software available for the three military architectures is relatively weak. Considering how easily modern microprogrammable processor hardware may be adapted to a given instruction-set architecture, there appears to be little reason to continue to use little-known or immature developments in future military computer systems.

The PDP-11 is one of the most successful architectures, in terms of user acceptance, in the history of the computer industry. It has been manufactured in the tens of thousands, and is widely used in almost every sort of OEM application. An extensive support software base exists for it, and DEC will continue to develop and support the architecture for the foreseeable future. It is clearly a satisfactory choice for the Military Computer Family. With the MCF intelligently defined and implemented, it will make available a family of militarized processors with excellent software development tools, and the capability to develop and maintain software on less expensive commercial equipment. This in turn will result in substantial cost and quality benefits in the application of computers to military systems.

8. Appendix

Table of Contents of The Final Report of the CFA Selection Committee the life cycle costs of the CFA finalists and describes two life cycle models (top-down and bottom-up) and the results of applying the methodology to those two models.

A-7 Volume VII - CFA/Software Licensing Discussions with the Three CFA Finalists (For Official Use Only)

Volume VII addresses the technical, financial, and legal issues arising out of discussions with the owner/manufacturer of each candidate computer architecture and describes the outcome of these discussions.

A-8 Volume VIII - CFA Final Selection

Volume VIII discusses the consideration by the Selection Committee of the results of the architecture evaluations described in Volumes II through VII of this report. The influencessthat the various results had on the final selection are described.

A-9 Volume IX - A Consideration of Issues in the Selection of a Computer Family Architecture

Volume IX addresses questions and controversial issues regarding the CFA Selection process that arose from both within and without the Selection Committee during the course of the CFA effort.

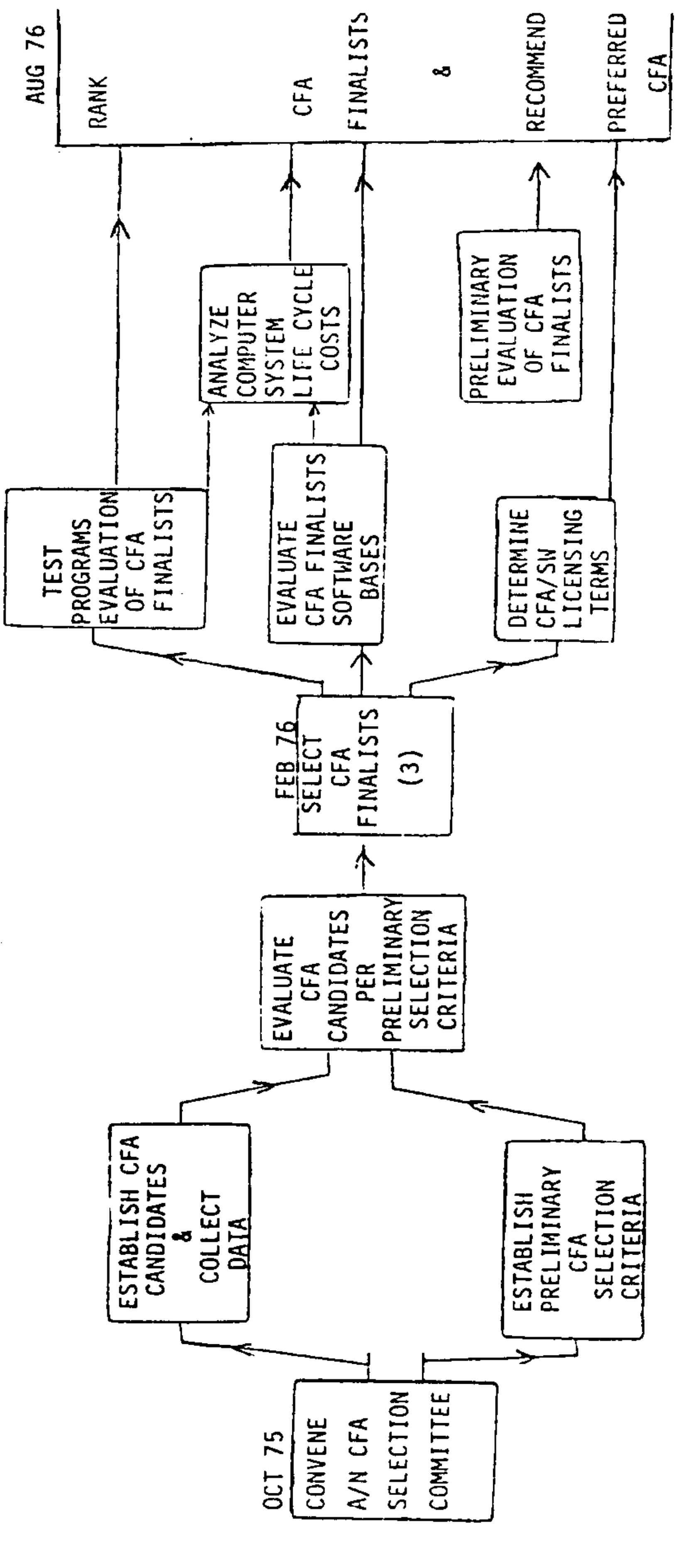
OCT 75
CONVENE A/N CFA SELECTION COMMITTEE
ESTABLISH CFA CANDIDATES & COLLECT DATA
ESTABLISH PRELIMINARY CFA SELECTION CRITERIA
EVALUATE CFA CANDIDATES PER PRELIMINARY SELECTION CRITERIA
FEB 76
SELECT CFA FINALISTS (3)
TEST PROGRAMS EVALUATION OF CFA FINALISTS
EVALUATE CFA FINALISTS SOFTWARE BASES
DETERMINE CFA/SW LICENSING TERMS
ANALYZE COMPUTER SYSTEM LIFE CYCLE COSTS
PRELIMINARY EVALUATION OF CFA FINALISTS
AUG 76
RANK CFA FINALISTS & RECOMMEND PREFERRED CFA

FIGURE 1. CFA Selection Procedure

Table 1 - Absolute Criteria for CFA Evaluation

(1) *Virtual Memory Support.*- The architecture must support a virtual to physical translation mechanism.

(2) *Protection.*- The architecture must have the capability to add new, experimental (i.e., not fully debugged) programs that may include I/O without endangering reliable operaion of existing programs.

(3) *Floating Point Support.*- The architecture must explicitly support one or more floating point data types with at least one of the formats yielding more than 10 decimal digits of significance in the mantissa.

(4) *Interrupts and Traps.*- It must be possible to write a trap handler that is capable of executing a procedure to respond to any trap condition and then resume operation of the program. The architecture must be defined such that it is capable of resuming execution following any interrupt.

(5) *Subsetability.*- At least the following components of an architecture must be able to be factored out of the full architecture:

Virtual-to-Physical Address Translation Mechanism

Floating Point Instructions and Registers (if separate from general purpose registers)

Decimal Instructions Set (if present in full architecture)

Protection Mechanism

(6) *Multiprocessor Support.*- The architecture must allow for multiprocessor configurations. Specifically, it must support some form of "test-and- set" instruction to allow the implementation of synchronization functions such as P and V.

(7)*Controllability of I/O.*- A processor must be able to exercise control over any I/O Processor and/or I/O Controller.

(8)*Extendibility.*- The architecture must have some method for adding instructions to the architecture consistent with existing formats. There must be at least one undefined code point in the existing opcode space of the instruction formats.

(9)*Read Only Code.*- The architecture must allow programs to be kept in a read-only section of primary memory.

Table 1 (cont.) - Quantitative Criteria for CFA Evaluation

(1) ***Virtual Address Space***

(a) V_1: The size of the virtual address space in bits.

(b) V_2: Number of addressable units in the virtual address space.

(2) ***Physical Address Space***

(a) P_1: The size of the physical address space in bits.

(b) P_2: The number of addressable units in the physical address space.

(3) ***Fraction of Instruction Space Unassigned***

(4) ***Size of Central Processor State***

(a) C_s2: The number of bits in the processor state of the full

(b) C_s2: The number of bits in the processor state of the minimum subset of the architecture (i.e., without Floating Point, Decimal, Protection, or Address Translation Registers).

(c) C_m1: The number of bits that must be transferred between the processor and primary memory to first save the processor state of the full architecture upon interruption and then restore the processor state prior to resumption.

(d) C_m2: The measure analogous to C_m1 for the minimum subset of the architecture.

(5) ***Virtualizability***

K: is unity if the architecture is virtualizable as defined in [PopG74] otherwise K is zero.

(6) ***Usage Base***

(a) B_1: Number of computers delivered as of the latest date for which data exists prior to 1 June 1976.

(b) B_2: Total dollar value of the installed computer base as of the latest date for which data exists prior to 1 June 1976.

(7) ***I/O Initiation***

I: The minimum number of bits which must be transferred between main memory and any processor (central, or I/O) in order to output one 8-bit to a standard peripheral device.

(8) *Direct Instruction Addressability*

D: The maximum number of bits of primary memory wwich one instruction can directly address given a single base register which may be used but not modified.

(9) *Maximum Interrupt Latency*

Let L be the maximum number of bits which may need to be transferred between memory and any processor (CP, IOC, etc.) between the time an interrupt is requested and the time that the computer starts processing that interrupt (given that interrupts are enabled).

Table 2 - Candidate Scores on Absolute and Quantitative Criteria

Architecture	Quantitative Criteria Score	Absolute Criteria Score
8/32	1.68 (Best)	Problem with interrupts and traps
PDP-11	1.43	Passed all
S/370	1.36	Passed all
AN/GYK-12	.94	Failed floating-point
ROLM/NOVA	.92	Failed virtual memory mapping and interrupts/traps
B6700	.91	Failed protection
SEL-32	.86	Failed virtual memory mapping
AN/UYK-7	.46	Failed floating point
AN/UYK-20	.44 (worst)	Failed protection

Table 3 - Test Programs

1. *I/O kernel, four priority levels,* requires the processor to field interrupts from four devices, each of which has its own priority level. While one device is being processed, interrupts from higher priority devices are allowed.

2. *I/O kernel, FIFO processing,* also fields interrupts from four devices, but without consideration of priority level. Instead, each interrupt causes a request for processing to be queued; requests are processed in FIFO order. While a request is being processed, interrupts from other devices are allowed.

3. *I/O device handler,* processes application programs' requests for I/O block transfers on a typical tape drive, and returns the status of the transfer upon completion.

4. *Large FFT,* computes the fast Fourier transform of a large vector of 32 bit floating point numbers. This benchmark exercises the machine's floating point instructions, but principally tests its ability to manage a large address space.

5. *Character search,* searches a potentially large character string for the first occurrence of a potentially large argument string. It exercises the ability to move through character strings sequentially.

6. *Bit test, set, or reset* tests the initial value of a bit within a bit string, then optionally sets or resets the bit. It tests one kind of bit manipulation.

7. *Runge-Kutta integration* numerically integrates a simple differential equation using third-order Runge-Kutta integration. It tests floating-point arithmetic.

8. *Linked list insertion* inserts a new entry in a doubly-linked list. It tests pointer manipulation.

9. *Quicksort* sorts a potentially large vector of fixed-length strings using the Quicksort algorithm. Like FFT, it tests the ability to manipulate a large address space, but it also tests the ability of the machine to support recursive routines.

10. *ASCII to floating point* converts an ASCII string to a floating point number. It exercises character-to-numeric conversion.

11. *Boolean matrix transpose* transposes a square, tightly-packed bit matrix. It tests the ability to sequence through bit vectors by arbitrary increments.

12. *Virtual memory space exchange* changes the virtual memory mapping context of the processor.

Table 4 - Test Program Experiment Results

Architecture	S	M	R
Interdata 8/32	.83	.85	.83
PDP-11	1.00	.93	.94
IBM S/370	1.21	1.27	1.29

Table 5 - Menu of Required Software Tool Types

Compilers
Macro Assemblers
Interactive Source Language Editors
Interactive Symbolic Debuggers
Extended Overlay Linker
Test Case Design Advisors
Integrated Library
Text Processing System
Data Base Management System
GP System Simulator
Time Sharing Operating System (TSOS) + VMM
Language Independent Monitors
Test Data Generator
Non-Interactive Symbolic Debugger
Computer System Simulator
Batch Source Language Editors
Language Dependent Monitors
TSOS + MPOS + VMM
Basic Assembler
RTOS + TSOS
Test Instrumenters & Analyzers
Automatic SW Production & Test
Basic Linker
Standards Enforcers
Reformatters
Test Data Auditor
Simple Overlay Linker
Data Base Design Aid

Table 6 - Tactical Support Software Base Evaluation

Architecture	Estimated Value of Current SSW Base	Estimated Cost To Eliminate Deficiency
8/32	$15.3 M	$25.9 M
PDP-11	$22.2 M	$19.1 M
S/370	$32.3 M	$ 9.6 M

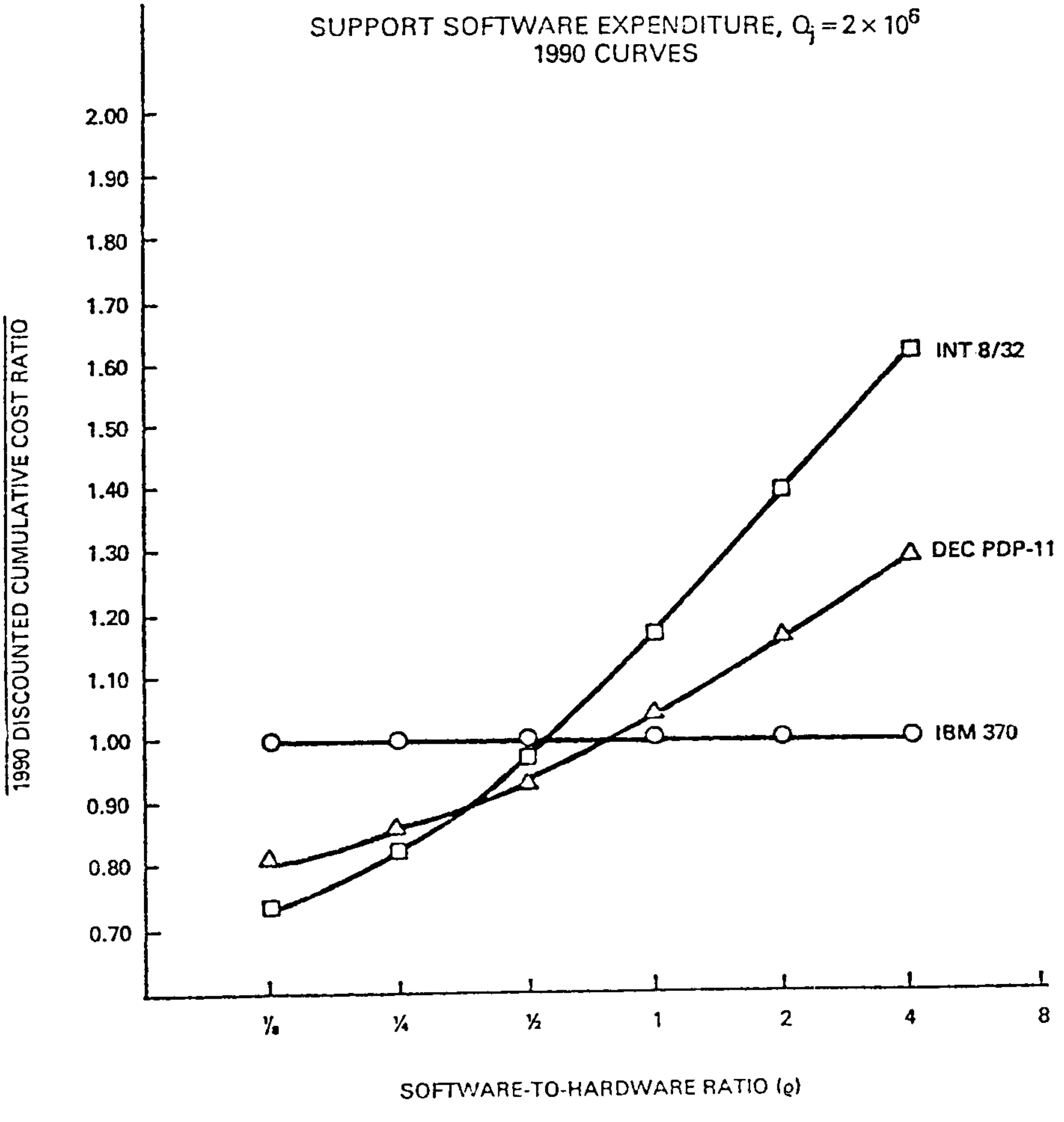

FIGURE 2. Top Down Life Cycle Cost Curves

A. AVERAGE TOTAL LIFE CYCLE COSTS ($000,000)

Type Cost	1976	1985
Hardware	$1750	$175
Software	162	75
TOTAL	$1912	$250

B. 1976 ARCHITECTURE COMPARISON

Architecture	# System Preferences	Relative Total Cost*		
		HDW	SW	Total
8/32	1	.92	1.33	.96
PDP-11	11	.91	1.00	.96
S/370	3	1.16	.67	1.12

C. 1985 ARCHITECTURE COMPARISON

Architecture	# System Preferences	Relative Total Cost*		
		HDW	SW	Total
8/32	-	.92	1.20	1.00
PDP-11	14.5	.91	.91	.91
S/370	0.5	1.16	1.09	1.09

* with respect to average cost; 1.00 equals average cost

TABLE 7. Summary: Bottom Up Life Cycle Cost Analysis

INITIAL SELECTION AND SCREENING OF THE CFA CANDIDATE COMPUTER ARCHITECTURES

Samuel H. Fuller,
Carnegie-Mellon University and
Naval Research Laboratory

Harold S. Stone,
University of Massachusetts

and

William E. Burr
US Army Electronics Command

TABLE OF CONTENTS

ABSTRACT

The initial selection criteria that were developed and used by the Army/Navy Computer Family Architecture (CFA) committee in their evaluation of alternative computer architectures is presented in this article. These initial criteria were used in this first phase of the CFA evaluation process to reduce the number of computer architectures from the original set of nine to the most promising three or four architectures for the more intensive evaluation discussed elsewhere [FulS77; WagJ77; SmiW77]. The machines selected by this initial ranking and screening process for further evaluation were the Interdata 8/32, DEC PDP-11, and the IBM S/370.

1. Introduction

The CFA selection committee was concerned with selecting a computer architecture to use in future military (ruggedized) computers and hence wanted to evaluate the merits of the computer architecture independent of any features, or flaws, of existing implementations of the computer. For this reason, the following definition of computer architecture was used by the CFA committee:

> Computer Architecture: The structure of the computer a programmer needs to know in order to write any machine-language program that will run correctly on the computer.

With a well specified architecture, details of data bus width, technology (core memory versus semiconductor memory, TTL versus ECL circuits), implementation speedup techniques, physical size of computer, etc. need not be of concern to the programmer and hence are not a part of the architecture. This separation of architecture and implementation is not a radically new idea [AmdG64]. The IBM System/360-370 series, the DEC PDP-11 series, and the Data General NOVA series are just three examples of where this has already been successfully accomplished to a greater or lesser degree.

This article first describes how the CFA selection committee chose the initial candidate architectures for evaluation, and then describes the criteria, the methodology, and the data used in ranking these architectures during the preliminary screening phase of the CFA project. At the point this procedure was formulated, it was known that time and money limitations would preclude doing a detailed analysis on all nine candidates; consequently an initial screening was necessary to limit the field to the three or four "best" candidates that would be subjected to a much more detailed analysis. This more detailed analysis, based on test programs, the support software bases of the architectures, and life cycle cost models is discussed in the accompanying articles.

Many detailed questions arose during the evaluation of these nine initial candidate architectures. It is impossible to review all these questions in this article, but we will discuss here the most important questions that arose, and interested readers are encouraged to refer to Volume II of the final report of the CFA committee for a detailed presentation of how and why each candidate architecture was evaluated as it was [FulS76a].

The mechanism for choosing the nine initial candidate architectures is discussed in the next section. The third and fourth sections then describe the nine absolute and seventeen quantitative criteria, respectively, and show how each of the candidate architectures was ranked on these criteria. The fifth section describes how the CFA committee combined the scores of the candidate architectures for each individual criteria to form a single, composite score for each architecture that reflected the relative importance of the seventeen quantitative criteria.

2. Initial Selection of Candidate Computer Architectures

The CFA selection process was initiated in March and April of 1975 when letters were sent to 35 Army and Navy organizations soliciting proposals for candidate computer architectures. As a result of these letters, and discussions at the first two CFA meetings, the following set of nine computer architectures was chosen:

Burroughs 6700
DEC PDP-11
IBM System/370
Interdata 8/32
Litton AN/GYK-12
ROLM Corporation 1664 (AN/UYK-28) *
SEL 32
Univac AN/UYK-7
Univac AN/UYK-20

There were on the order of 100 viable computer architectures in 1975 that might have been considered by the CFA committee for selection [GMLC75]. The decision as to what set of architectures would be evaluated remained open from March through December of 1975. The nine architectures listed above were selected for evaluation because they met two essential criteria: (1) the CFA committee agreed the architecture might be a reasonable choice for future military computers and (2) there was a CFA committee member sufficiently convinced of the value of the computer architecture that he was willing to act as its advocate in the subsequent evaluation phase.

3. Absolute Criteria

The CFA selection committee specified nine absolute criteria that they felt a candidate computer architecture needs to satisfy if it is going to meet the

* The AN/UYK-28 is instruction-set upward-compatible with the Data General NOVA computer architecture. Other ROLM computers that are also compatible with the NOVA architecture are the AN/UYK-19 and AN/UYK-27. The AN/UYK-28 is incompatible with the Data General ECLIPSE computer architecture, Data General's upward-compatible extension of the NOVA.

requirements of future military computer systems. All the absolute criteria (with the exception of the subsetability criterion) had to be satisfied by an implementation of the architecture which was operational by 1 January 1976. This eliminated speculative decisions based on promises or potential solutions that looked inviting, but might not come to fruition. Failure to satisfy any absolute criterion resulted in the elimination of the architecture from further consideration. The nine absolute criteria are given below. The formal statement of each criterion is underlined, while explanations and examples are not underlined. Many of the comments that follow the definition of an absolute criteria are the result of the experience gained when the CFA committee evaluated the nine candidate architectures against these criteria [StoH76]. Table 3-1 shows which absolute criteria each candidate architecture passed or failed.

Virtual Memory Support.-The architecture must support a virtual to physical address translation mechanism.

The intent of this criterion is to take advantage of the widely used feature of many machines that is known as virtual memory. Many advantages accrue to architectures that support virtual address translation mechanisms, the most notable of which is the ability to simplify programming by freeing the programmer of explicit management of his primary memory and providing a mechanism for keeping only the active portions of a program in high-speed memory.

The answers for this criterion listed in Table 3-1 are not controversial, except for the AN/UYK-20. This architecture provides the page registers necessary for relocation, but does not limit the ability to change these registers to privileged programs. Some members of the CFA committee felt that preventing user state access to the page registers was a necessary aspect of virtual memory; others disagreed.

The full CFA committee voted to fail the AN/UYK-20 on this criteria. The ROLM 1664 and SEL 32 both failed this criterion because each of these architectures provide a mechanism commonly known as "bank switching", which the committee felt was not an adequate memory translation mechanism.

Protection.-<u>The architecture must have the capability to add new, experimental (i.e., not fully debugged) programs that may include I/O without endangering reliable operation of existing programs</u>. The intent of this criterion is to provide a mechanism in the hardware for aiding software development, and for preventing certain catastrophic software failures from occurring in the field. Architectures that use a privileged mode to protect vital registers and system resources generally meet this criterion.

The AN/UYK-20 failed this criterion because it lacks memory protection; any user can modify the contents of the relocation registers, and thereby read and write any word in memory. Another generic way for an architecture to fail the protection criterion is for a program to have the ability to put the machine into a noninterruptable state for an indefinite time. Architectures that permitted nonterminating instructions were carefully examined to identify if these were, or were not, interruptable.

Floating-Point Support.-<u>The architecture must explicitly support one or more floating-point data types with at least one of the formats yielding more than 10 decimal digits of significance in the mantissa</u>. The significance measure was determined as representative of the most stringent requirements actually encountered.

The AN/GYK-12 failed this criterion because it does not support floating point operations. The AN/UYK-7 failed because it supports a single, 64-bit floating point format with only 31 bits (9.2 decimal digits) of mantissa. Because this is so close to the borderline, one might reconsider requirements on significance to determine how firm the 10 decimal digit criterion is. (Had the AN/UYK-7 looked like an otherwise excellent architecture, it is likely that the committee would have relaxed the floating point absolute criterion for it.)

Interrupts and Traps.-It must be possible to write a trap handler that is capable of executing a procedure to respond to any trap condition and then resume operation of the program.

For example, if the processor receives a page-fault trap from the address translation unit, it must be able to request the appropriate page be brought in from secondary storage and then resume execution. If resumption of execution is logically impossible (e.g., an attempt to store an operand into a read-only segment of virtual memory or fetch an instruction with a parity error) then the trap handler should be able to abort the program with an indicator of which instruction and/or operand caused the termination.

A similar requirement exists for interrupts: the architecture must be defined such that it is capable of resuming execution following any interrupt (e.g., power failure, disk read error, console halt).

Another intent of this criterion is to permit extensions and subsets of an architecture to operate correctly so programs can be upward or downward compatible. The subsets and extensions may differ drastically in size, cost, and performance, but every program written for the native architecture can run on the subset or extended machine.

The Interdata 8/32 had difficulty satisfying this criterion since it has variable length instructions, and there is no way after a trap or an interrupt to tell whether the instruction which was being executed was a 16, 32, or 48 bit instruction. This may be a problem when it is desirable to correct the cause of the fault, and then re-execute (or resume) the instruction. Due to uncertainties in the definition of the Interdata 8/32 architecture, the CFA committee was not able to resolve whether or not the Interdata 8/32 satisfied this criterion.

Subsotability.-At least the following components of an architecture must be able to be factored out of the full architecture:

a. Virtual-to-Physical Address Translation Mechanism

b. Floating Point Instructions and Registers (if separate from general purpose registers)

c. Decimal Instructions Set (if present in full architecture)

d. Protection Mechanism

Implementations of the architectures on small machines for dedicated applications must not be required to include features of the architecture intended for use on larger, multiprogrammed, multi-application configurations. Existence of such subsets did not have to be demonstrated in an operational implementation of the architecture.

Because there was no operational method for testing subsetability, we could not challenge positive replies for any of the nine candidate architectures. However, the B-6700 and the AN/UYK-7 have not been subsetted in the sense of the criterion, so that their subsetability is more speculative.

In order to retain program compatibility across the implementations of the architecture, this criterion was extended to include the following requirement: The trap mechanism of the architecture must be defined such that instructions in the full architecture, but not implemented in the subset machine, trap on the subset machine and that it be possible to write trap routines for the subset machine that allow it to interpretively execute those instructions not implemented directly in hardware (or firmware) and then resume execution. (This is an elaboration of absolute criterion 4.)

Multiprocessor Support.-The architecture must support some form of "test-and-set" instruction to allow for the communication and synchronization of multiple processors.

The intent of this criterion is to be sure that the basic architecture can support multiprocessor configurations.

Input/Output Controllability.-A processor must be able to exercise absolute control over any I/O processor and/or I/O controller.

The interpretation of the criterion proved rather difficult. While all architectures necessarily permitted individual devices to be started and queried for status, there were varying degrees of control exercisable with respect to stopping the devices. It is reasonable to stop all input/output, or to stop selected devices. All architectures had some way of stopping a single device and stopping all devices, but how they did it varied widely in efficiency.

Extensibility.-The architecture must have some method for adding instructions to the architecture consistent with existing formats. There must be at least one undefined code point in the existing opcode space of the instruction formats. All nine candidate architectures have unused instructions, so all passed this criterion.

Read-Only Code.-It must be possible to execute programs from read-only storage.

This criterion is intended to permit an added degree of reliability by permitting programs to be stored in a nonvolatile read-only memory. However, a program can be rewritten to be read-only on any of the nine architectures, even if that architecture does not support special types of instructions to facilitate this. It might have been more meaningful to examine this question quantitatively.

Table 3-1 shows the score of each candidate architecture on each of the absolute criteria. Note that none of the nine architectures failed to meet the last five criteria: subsetability, multiprocessor support, I/O controllability, extensibility, and read-only code. This is in part the case because we limited our evaluation to reasonably successful architectures, but is partly the result of not defining these criteria precisely enough prior to applying them to the candidate architectures. For example, by not clearly defining how to test for the practical subsetability of an architecture, we made it virtually impossible for an architecture to fail this criteria. Subsequent studies would be well advised to consider more precise definitions of these (and any additional) absolute criteria before evaluating alternative architectures against them.

4. Quantitative Criteria

In addition to the absolute criteria, the CFA committee specified seventeen quantitative criteria that they felt would be helpful in the initial screening process. A number of these quantitative criteria measure attributes of a computer architecture better measured by benchmarks, or test programs [FulS77a]. However, the CFA committee recognized that it did not have the resources to run benchmarks on all nine candidate architectures and therefore proceeded with the use of these quantitative criteria to help select three or four candidate architectures, out of the original nine candidate architectures, for more intensive study via test programs.

The quantitative criteria are described below and the score of each architecture on the quantitative criteria is given in Table 4-1.

Virtual Address Space.-

V_1: The size of the virtual address space in bits.

V_2: Number of addressable units in the virtual address space.

Two aspects of these measures were open to interpretation. The CFA committee settled on the following interpretation for treating bank switching: the virtual address for a machine with bank switching is the address within a bank. The effect of bank switching is to increase the size of the physical rather than the virtual address.

The second interpretation centered on the notion of "addressable unit". There are several degrees of addressability. An item may be fully addressable in the sense that it can be accessed by the address produced by an effective address computation. The committee also decided, however, that instructions such as the IBM S/370 Test Under mask, and the OR Immediate allowed the testing and setting of individual bits, and provided a minimum addressable unit of 1 bit.

Physical Address Space.-

P_1: The size of the physical address space in bits.

P_2: The number of addressable units in the physical address space.

Where bank switching has been implemented, the physical address measures include all the banks of memory available. For computers with virtual address translation, the physical address is the address resulting from the virtual-to-physical address translation. The physical address space is defined apart from any implementation, since the physical address space size is defined by the effective address calculation process or the virtual address translation process and need not be equal to the largest memory configuration yet delivered.

Fraction of Instruction Space Unassigned.-It is important to select an architecture that will allow reasonable growth over its expected lifetime. Let U be defined as the fraction of the instruction space in the architecture that is unassigned. Specifically:

$$U = \sum_{1 \leq i < \infty} u_i . 2^{-i} \qquad (4.1)$$

where u_i is the number of unassigned instructions of length i.

Size of Control Processor State.-The amount of information that must be stored or loaded upon interrupt and/or context swapping is clearly an important factor in the response of real time systems and in the overhead of multiprogramming systems. Let the processor state be defined as all the bits of information in a processor that must be saved in order to be able to restart an interrupted process at a later date. Processor states normally include the accumulators, index registers, program counter, condition codes, memory mapping registers, interrupt mask registers, etc.

C_{s1}: <u>The number of bits in the processor state of the full architecture</u>.

C_{s2}: <u>The number of bits in the processor state of the minimum subset of the architecture</u> (i.e., without Floating Point, Decimal, Protection, or Address Translation Registers).

C_{m1}: <u>The number of bits that must be transferred between the processor and primary memory to first save the processor state of the full architecture upon interruption and then restore the processor state prior to resumption</u>. This measure differs from C_{s1} above in that "register bank switching", where provided for in the candidate architectures, may eliminate the need to save some registers in primary memory, while the instruction fetches required to save the state are included in C_{m1} but not in C_{s1}.

C_{m2}: The measure analogous to Cm1 for the minimum subset of the architecture.

These measures give an approximation to the complexity of the implementation of the architectures, as well as a measure of the responsiveness of the architectures to worst-case context changes for interrupt processing.

If an architecture provides for several sets of certain registers to provide fast switching or multiple contexts, and if a program uses only one such register set when it runs in one context, then only one set of these registers is used in calculating C_{s1}.

Usage Base.-

B_1: Number of computers delivered as of the latest date for which data exists prior to 1 June 1976.

B_2: Total dollar value of the installed computer base as of the latest date for which data exists prior to 1 June 1976.

These two measures are meant to be approximate indicators of the existing software and programmer experience base. A single individual determined the value of these measures for all candidate architectures from standard sources.

I/O Initiation.-

I: The minimum number of bits which must be transferred between main memory and any processor (central, or I/O) in order to output one 8-bit byte to a standard peripheral device.

Although this measure was intended to give some insight into the responsiveness of an architecture, it is very difficult to construct an interpretation of the measure that serves this purpose well. The measure counts relatively few bits for some architectures, and this, in turn, makes the measure very sensitive to changes of a few bits. The I measure is also sensitive to several assumptions about exactly what actions are to be performed in doing the input/output operation, and where parameters for the operation are found. Unfortunately, this sensitivity made the I measure very arbitrary, and a rather inexact measure of input/output responsiveness. The precise, and somewhat lengthy, definition of I is given in [FulS76a].

Virtualizability.-

K: is unity if the architecture is virtualizable as defined in [PopG74], otherwise, K is zero.

The intent of this criterion is to capture the concept of virtual machines that has been used to advantage in some commercial computer systems (e.g., IBM's VM/370). An architecture that supports virtual machines provides a mechanism for a privileged, stand-alone program to run as an unprivileged task and produce the results identical to those it produces as a privileged program. The importance of this idea is that an operating system can be run in user mode as a subsystem of another operating system.

The definition of virtual machine as provide by Popek and Goldberg in their article in CACM [PopG74] is a very strict definition that guarantees that any operating system that can run stand-alone on architecture X, can also run on architecture X in nonprivileged mode. If an architecture fails this definition it may still support virtual machines in a more limited sense.

Direct Instruction Addressability.-

D: The maximum number of bits of primary memory which one instruction can directly address given a single base register, which may be used but not modified.

Large displacement fields in instructions generally simplify programming because they reduce the need to set base registers and to maintain addressability. Because an architecture may have several different instruction formats, each with different displacement field formats, the committee required that the format selected for this measure be the one used for standard LOAD and STORE operations, or the equivalent thereof. This eliminated anomalies, like the MOVE CHARACTER LONG in the IBM S/370 architecture, from consideration.

Maximum Interrupt Latency.-Let L be the maximum number of bits which may need to be transferred between memory and any processor (central processor, I/O controller, etc.) between the time an interrupt is requested and the time that the computer starts processing that interrupt (given that interrupts are enabled). This may be interpreted as a measure of the longest non-interruptable instruction or sequence of instructions. Architectures with nonterminating non-interruptable instructions have infinite L measures and are so indicated in Table 4-1.

Subroutine Linkage.-

J_1: The number of bits which must be transferred between the processor and memory to save the user state, transfer to the called routine, restore the user state, and return to the calling routine, for the full architecture. No parameters are passed.

J_2: The analogous measure to S1 above for the minimum architecture (e.g., without Floating Point registers).

This measure gives an indication of the size of overhead that might be encountered in doing subroutine calls in the worst case for the biggest and smallest machines in the family. The bits counted here are related to the count in CS_1, CS_2, CM_1, and CM_2. By presumption, the bits that are stored for J_1 are exactly those for CS_1, except that it is not necessary to save the protection registers, memory map registers, interrupt mask, and other registers that determine the global context for a program. Architectures with small processor states or that have LOAD/STORE MULTIPLE instructions show up well on these measures.

5. Composite Score of the Quantitative Criteria

After applying the quantitative criteria just discussed, the CFA committee had to determine how the performance of the candidate architectures on these criteria would be used to screen out all but three or four of the architectures for further consideration in the test program and software evaluation phases of the study. Clearly, the candidate architectures should be ordered relative to each of the seventeen quantitative criteria and these independent orderings studied to detect weaknesses and strengths of the competing architectures. However, some summary measure was ultimately needed to assist the committee in its selection of the final architectures to undergo more intensive study. A variety of thresholding and weighing schemes were proposed, but the particular scheme that follows was the scheme chosen by the CFA committee.

5.1. Relative Weighing of Criteria

Each voting organization of the CFA committee was given 100 points to distribute among the various measures to indicate their relative importance to the organization. The weight for criterion x, W[x], was defined as the total number of points given criterion x by all the voting CFA organizations, divided by the total number of points handed out. The weights for the quantitative criteria based on responses from 24 voting CFA committee members is given in Table 5-1.

5.2. Normalization

When attempting to combine these quantitative measures into a composite measure we faced two problems:

a. The measures are defined such that good computer architectures maximize some measures and minimize others. Specifically, the measures that a computer architecture should maximize are: V_1, V_2, P_1, P_2, U, K, B_1, B_2, and D; while the measures that should be minimized are: C_1, C_2, C_3, C_4, I, L, J_1, and J_2.

Let our composite measure be a maximal measure and transform all minimal measures to maximal measures by taking the reciprocal: $X' = 1/X$.

b. Measures that inherently involve large magnitudes are not necessarily more important than smaller measures. For example, V_1 is on the order of 10^4 to 10^9 while K is either 0 or 1.

To resolve this problem of differing scale, the values for the quantitative criteria were normalized by dividing each value by the average value of the criterion over the set of nine architectures. For example, the nine measures for criteria I are (64, 16, 48, 16, 128, 64, 169, 80, 32), the average value is 68.6, and the normalized measures are (0.93, 0.23, 0.70, 0.23, 1.87, 0.93, 2.47, 1.17, 0.47).

Normalized measures have the attractive properties that they all lie in the range (0,M); have a mean across the set of M architectures of unity; and the standard deviation of the set of normalized measures is in the interval (0, $M^{0.5}$). We could have taken the normalization process a step further and adjusted the spread of each measure so that the measure gave a standard deviation of unity (or some other constant) across the set of architectures being evaluated. We did not do this for all measures. Some measures were better "discrimination functions" than others and we did not want in general to lose this information by further normalization. However, the committee agreed that it is important to normalize the standard deviation of some of the measures; specifically, V_1, V_2, P_1, P_2 and D were normalized to have a mean and standard deviation of unity. These measures may differ by several orders of magnitude between candidate architectures, but the CFA Committee did not feel that the utility, as expressed by the measures, differ by orders of magnitude.

5.3. Scaling and Composition of the Quantitative Measures

In order to combine the individual measures the committee used a simple, linear sum of each normalized measure X scaled by its corresponding weighing coefficient W[X]. The weighing coefficients have been defined so that they sum to unity and hence the composite measure A is in fact a normalized measure with a mean of 1. Using the weights given in Table 5-1 and the values of the quantitative criteria given in Table 4-1, we get the composite measures for the candidate architectures shown in Table 5-2.

There was some valid concern by members of the CFA committee about the role of the weighing of the measures, the normalization of the measures, and the measures themselves in the selection of finalists. However, upon detailed examination of the results we found that, given the weights applied by the committee as an indication of the importance of idealized concepts, the finalists selected are very insensitive to the exact details of the selection procedure. Almost any reasonable methodology for measuring the key concepts quantitatively would select the same finalists.

6. Summary

This article has presented the nine absolute criteria and the seventeen quantitative criteria used by the CFA committee in their initial screening on the initial candidate computer architectures. The scores for each of the candidate architectures are given in Tables 3-1 and 4-1 for the absolute and quantitative criteria, respectively. Only the IBM S/370 and PDP-11 architectures passed all the absolute criteria. The Interdata 8/32 architecture is not well defined with respect to trap handling and there remains some question as to whether it meets the requirements of the interrupt and trap handling criteria. The remaining six candidate architectures failed one or more of the absolute criteria specified by the CFA committee. A weighing scheme was developed by the CFA committee for the quantitative criteria and the composite scores of the nine candidate architectures are given in Table 5-2. The quantitative criteria showed that the Interdata 8/32, PDP-11, and IBM S/370 lead the other architectures by comfortable margins. These results were used by the CFA committee to reduce the field of candidate architectures to three finalists -- the IBM S/370, the PDP-11, and the Interdata 8/32 -- for more thorough evaluation.

This article has indicated some of the areas where we had difficulty applying the criteria and the final report of the CFA committee goes into these difficulties, and their resolution, in much greater detail [FulS76a]. The fact remains, however, that if we had to compare a set of computer architectures again, we would need to go through a similar "initial screening" process; it is just too costly and time-consuming to expect to be able to evaluate more than a small set of architectures via any more comprehensive means such as benchmarking. The absolute and quantitative criteria used by the CFA committee have the attractive property that they can be determined directly from the definition of the computer architecture (or from a survey of computer installations for criteria B_1 and B_2). Reflecting back on the history of the CFA project, we estimate that it took from two to five man-days to evaluate each of the computer architectures against the criteria discussed in this article, plus a two day meeting of the entire CFA committee to resolve differences of interpretation, and it took from six to nine man-months to evaluate each of the computer architectures via the set of test programs, support software evaluation, and life cycle cost models in the subsequent stages of the CFA project.

Acknowledgements

The criteria and methodology described in this article benefited from the ideas and criticism of many members of the CFA committee. R. Estell, L. Haynes, and N. Tinkelpaugh, as members of the selection criteria subcommittee, each made important contributions to the formulation of the initial screening process.

		CANDIDATE COMPUTER ARCHITECTURES								
	ABSOLUTE CRITERION	IBM S/370	INTER-DATA 8/32	ROLM	DEC PDP-11	UNIVAC AN/UYK-7	SEL 32	BURROUGHS B6700	UNIVAC AN/UYK-20	LITTON AN/GYK-12
1	Virtual Memory	Y	Y	N	Y	Y	N	Y	N	Y
2	Protection	Y	Y	Y	Y	Y	Y?	N	N	Y?
3	Floating Point	Y	Y	Y	Y	N	Y	Y	Y	N
4	Interrupts/Traps	Y	?	Y	Y	Y	Y	Y	Y	Y
5	Subsettability	Y	Y	Y	Y	Y?	Y	Y?	Y	Y?
6	Multi Processor	Y	Y	Y	Y	Y	Y	Y	Y	Y
7	I/O Controllability	Y	Y	Y	Y	Y	Y	Y	Y	Y
8	Extensibility	Y	Y	Y	Y	Y	Y	Y	Y	Y
9	Read-Only Code	Y	Y	Y	Y	Y	Y	Y	Y	Y
	SUMMARY	Y	?	N	Y	N	N	N	N	N

Y	Yes, Meets Criteria
N	No, Fails Criteria
Y?	Yes (but with some reservations)
?	Unresolved

Table 3-1. Candidate Architecture Value for Absolute Criteria

#	QUANTI-TATIVE CRITERIA	CANDIDATE CFA's								
		IBM S/370	INTER-DATA 8/32	ROLM	DEC PDP-11	UNIVAC UYK-7	SEL 32	BURROUGHS B6700	UNIVAC UYK-20	LITTON GYK-12
1	V_1**	27	27	20	20	24	22	24	20	20
2	V_2**	27	27	20	19	24	22	20	17	20
3	P_1**	27	27	***22	25	23	***26	24	20	29
4	P_2**	27	27	***22	24	23	***26	20	17	29
5	U	.371	.355	.039	.043	.15	.450	.019	.125	.219
6	CS_1	1344	1632	1008	1168	992	304	306	1328	1008
7	CS_2	576	576	112	144	448	288	204	336	752
8	CM_1	3168	1120	1882	736	1472	768	408	2256	1344
9	CM_2	1312	32	544	480	1472	704	408	720	1088
10	K	1	0	0	1	0	0	0	0	0
11	B_1	17,300	185	****13,800	14,700	346	75	90	400	30
12	B_2*****	16,000	14	169	311	147	23	207	8	6
13	I	64	16	48	16	128	64	169	80	32
14	D	15	27	20	19	18	22	18	20	20
15	L	6192	560	114	112	2112	288	255	--	1376
16	J_1	1904	2368	1360	1040	1280	960	459	1408	1344
17	J_2	1136	1280	320	400	1280	960	459	640	1088

**These values are of the form 2^x where x = indicated data except for B6700 which is of the form $3(2^x)$.

With memory bank switching. * Includes Novas. *****In *X 10^6.

Table 4-1. Candidate CFA Values for Quantitative Criteria

CRITERION	ARMY WEIGHTS	NAVY WEIGHTS	FULL CFA COMMITTEE WEIGHTS
V_1	.0412	.0444	.0433
V_2	.0438	.0575	.0529
P1	.0425	.0706	.0612
P2	.0387	.0637	.0554
U	.0513	.0644	.0600
CS1	.0587	.0375	.0466
CS2	.0675	.0219	.0371
CM1	.0700	.0544	.0596
CS2	.0713	.0319	.0450
K	.0500	.0587	.0558
B1	.0450	.0244	.0313
B2	.0200	.0281	.0254
I	.0875	.1419	.1238
D	.0912	.1081	.1025
L	.0812	.0969	.0917
J1	.0637	.0626	.0629
J2	.0762	.0331	.0475

Table 5-1. Quantitative Criteria Composite Weights

Architecture	Score
Interdata 8/32	1.68
PDP-11	1.43
IBM S/370	1.36
AN/GYK-12	0.94
ROLM	0.92
B6700	0.91
SEL-32	0.86
AN/UYK-7	0.46
AN/UYK-20	0.44

Table 5-2. Ranking Based on the Quantitative Criteria

EVALUATION OF COMPUTER ARCHITECTURES VIA TEST PROGRAMS

Samuel H. Fuller
Carnegie-Mellon University and
Naval Research Laboratory

William E. Burr
U.S. Army Electronics Command

Paul Shaman
Carnegie-Mellon University

and

David Lamb
Carnegie-Mellon University

TABLE OF CONTENTS

ABSTRACT

This article presents the evaluation of the Computer Family Architecture (CFA) candidate architectures via a set of test programs. The measures used to rank the computer architectures were S, the size of the test program, and M and R, two measures designed to estimate the principal components contributing to the nominal execution speed of the architecture. Descriptions of the twelve test programs and definitions of the S, M, and R measures are included here. The statistical design of the assignment of test programs to programmers is also discussed. Each program was coded from two to four. times on each machine to minimize the uncertainty due to programmer variability. The final results show that for all three measures (S, M, and R) the Interdata 8/32 is the superior architecture, followed closely by the PDP-11, and the IBM S/370 trailed by a significant margin.

1. Introduction

While there are many useful parameters of a computer architecture that can be determined directly from the principles of operation manual, the only method known to be a realistic, practical test of the quality of a computer architecture is to evaluate its performance against a set of benchmarks, or test programs. In a previous article [FulS77b], we presented a set of absolute and quantitative criteria that the CFA committee felt provided some indication of the quality of the candidate computer architectures. It is important to emphasize, however, that throughout the discussion of these criteria it was understood that a benchmarking phase would be needed, and that many of the quantitative criteria were being used to help construct a reasonable "prefilter" that would help to reduce the number of candidate computer architectures from the original nine to a final set of three or four. As described in the preceding article, this initial screening in fact reduced the set of candidate computer architectures to three: the IBM S/370, the PDP-11, and the Interdata 8/32.

The concept of writing benchmarks or test programs, is not a new idea in the field of computer performance evaluation and is generally considered the best test of a computer system [cf. LucH71; BerN75; WicB73]. For the purpose of the CFA committee, we define a <u>test program</u> to be a relatively small program (100 to 500 machine instructions) that was selected as representative of a class of programs. The CFA committee's test program evaluation study described here had to address the central problems facing conventional benchmarking studies:

a. How is a representative set of test programs selected?

b. Given limited manpower, how are programmers assigned to writing test programs in order to maximize the information that can be gained?

We faced an additional problem because we evaluated computer architectures, independent of any of their specific implementations. In other words, when evaluating particular computers, time is the natural measure of how fast a test program can be executed. However, a computer architecture does not specify the execution time of any instructions and so an alternative to time must be chosen as a metric of execution speed.

This article explains how the CFA committee addressed the above questions and presents the results of the test program evaluation of the three candidate architectures. The next section, Section 2, describes how the 12 test programs used in the evaluation process were selected. Section 3 explains the measures of architecture performance that were used in this study. Section 4 explains how 16 programmers were assigned from six to nine programs each, in order to get a set of slightly over 100 test program implementations that were used to compare the relative performance of the candidate architectures. The principle results of the test program evaluation are presented in Section 5 and Appendix A contains the actual S, M, and R measurements of all of the test programs. For the actual specifications of the test programs, details of the evaluation process beyond the scope of this article, and a chronology of the CFA test program study see [FulS76b].

2. Test Program Specification

2.1. Alternative Approaches

A number of alternative test program specifications were considered by the CFA committee. A tempting proposal was to use test programs written in a Higher-Order Language (HOL). This had the advantage of allowing a single HOL source program to be used for all the architectures to be tested. This also would have permitted the use of existing benchmark programs, which were available from several sources (FCDSSA, and NADC), and which were extracted from "real" military systems. One disadvantage of this approach was that no one language, even FORTRAN, was available on all the nine initial candidate architectures and those languages developed for use in tactical military applications (e.g., JOVIAL, CMS-2, CS-4, and TACPOL) were each available on only a few of the candidate architectures. There are FORTRAN IV and COBOL compilers available for each of the three final candidate architectures; however, neither FORTRAN nor COBOL are widely used in tactical military applications. The major disadvantage, however, was that there is no practical way to separate the effects of compiler quality from the effects of architectural efficiency, and the object of the test program study was to measure only the architecture. The results obtained from HOL test programs would necessarily involve a significant undetermined component, which would be due to variations in the efficiency of compilers that are unlikely to be extensively used in tactical military applications, and because these unmeasurable compiler effects might well mask genuine differences in the intrinsic efficiencies of the architectures.

Using standard (Machine-Oriented) assembly language for the test programs was the obvious alternative to the use of Higher Order languages, but it had several obvious disadvantages. First, each program would have to be recoded for each machine, adding to the effort involved. Moreover, this introduced programmer variability into the experiment, and previous studies have shown programmer variability to be large (variation of factors of 4:1 or more are commonly accepted). Finally, it is much more expensive to code in assembly language than in Higher Order Languages, and this would limit the size or number of the test programs. Nevertheless, the committee felt that there were ways to limit, separate, and measure these programmer effects, while there was no practical way to limit or separate the effects of compiler efficiency. It was therefore decided that the test programs would, of necessity, be coded in assembly language.

2.2. Guidelines for Test Programs Specification

The Test Program Subcommittee attempted to establish a strategy for defining and coding the test programs that would minimize the variability due to differences in programmer skill. The strategy devised was as follows:

a. The test programs would be small "kernel" type programs, of not more than 200 machine instructions. (In the end, a few test programs required more than 200 instructions.) It was felt that only small programs could be specified and controlled with sufficient precision to minimize the effects of programmer variability.

Moreover, resources were not available to define, code, test, and measure a significant set of larger programs.

b. The programs were defined as structured programs, using a PL/I-like Program Definition Language (PDL) and then "hand translated" into the assembly languages of the respective architectures.

c. Programmers were not permitted to make algorithmic improvements or modifications, but rather were required to translate the PDL descriptions into assembly language. Programmers were free to optimize their test programs to the extent possible with highly optimizing compilers. This "hand translation" of strictly defined algorithms was expected to reduce variations due to programmer skill.

d. All test programs except the I/O Interrupt test programs were coded as reentrant, position-independent (or self-relocating) subroutines. This was believed to be consistent with the best contemporary programming practice and provides a good test of an architecture's subroutine and addressing capabilities.

2.3. Selection of the Twelve Test Programs

The CFA committee appointed a subcommittee responsible for developing a set of test program specifications consistent with the guidelines just discussed. This subcommittee defined a set of 21 test programs that were intended to be broadly representative of the basic types of operations performed by military computer systems. The CFA committee reviewed these 21 test programs, committee members were asked to rank the relevance of these test programs to the applications of their particular organization, and it was agreed that the top 12 programs would be the basis of the test program study. (The rationale for using 12 test programs is explained in Section 4, where the statistical design of the test program assignments is presented.) The full specification of the 12 selected test programs is given in [FulS76b] and a brief description of these test programs is given below.

A. I/O kernel four priority levels, requires the processor to field interrupts from four devices, each of which has its own priority level. While one device is being processed, interrupts from higher priority devices are allowed.

B. I/O kernel, FIFO processing, also fields interrupts from four devices, but without consideration of priority level. Instead, each interrupt causes a request for processing to be queued; requests are processed in FIFO order. While a request is being processed, interrupts from other devices are allowed.

C. I/O device handler processes application programs' requests for I/O block transfers on a typical tape drive, and returns the status of the transfer upon completion.

D. Large FFT computes the fast Fourier transform of a large vector of 32-bit floating point complex numbers. This benchmark does exercise the machine's floating point instructions, but principally tests its ability to manage a large address space. (Up to one half of a million bytes may be required for the vector.)

E. Character search, searches a long character string for the first occurrence of a potentially large argument string. It exercises the ability to move through character strings sequentially.

F. Bit test, set, or reset tests the initial value of a bit within a bit string, then optionally sets or resets the bit. It tests one kind of bit manipulation.

G. Runge-Kutta integration numerically integrates a simple differential equation using third-order Runge-Kutta integration. It is primarily a test of floating-point arithmetic and iteration mechanisms.

H. Linked list insertion inserts a new entry in a doubly-linked list. It tests pointer manipulation.

I. Quicksort sorts a potentially large vector of fixed-length strings using the Quicksort algorithm. Like FFT, it tests the ability to manipulate a large address space, but it also tests the ability of the machine to support recursive routines.

J. ASCII to floating point converts an ASCII string to a floating point number. It exercises character-to-numeric conversion.

K. Boolean matrix transpose transposes a square, tightly- packed bit matrix. It tests the ability to sequence through bit vectors by arbitrary increments.

L. Virtual memory space exchange changes the virtual memory mapping context of the processor.

The specifications, written in the Program Definition Language, were intended to completely specify the algorithm to be used, but allow a programmer the freedom to implement the details of the program in whatever way best suited the architecture involved. For example, in the ASCII-to-floating-point benchmark, program J, the PDL specification included the statement:

> NUMBER ← integer equivalent of characters POSITION to J-1 of A1 where character J of A1 is "."

This description instructs the programmer to convert the character substring POSITION, POSITION +1,...,J-1, to an integer and store the result in the integer NUMBER. It left up to the programmer whether he would sequence through the string character-by-character, accumulating an integer number until he found a decimal point, or perhaps (on the S/370) use the Translate-and-Test (TRT) instruction to find the decimal point, and then use PACK and Convert-to-Binary (CVB) to do the conversion.. It did forbid him to accumulate the result as a floating point number directly, forcing him to first convert to an integer and then to floating point.

2.4. Procedures for Writing, Debugging, and Measuring the Test Programs

The test programs were written by seventeen programmers at various Army and Navy laboratories and at Carnegie-Mellon University. A set of reasonably comprehensive instructions and conventions were needed to insure that the various programmers produced results that could be compared in a meaningful way. Section 4 of this article discusses the assignments made to the programmers, and shows how these assignments were made to minimize the distortion of the final conclusions due to variations between programmers. In addition, we also agreed that it was not sufficient to just write the test programs in assembly language. We instructed each programmer that all of the test programs that he wrote had to be assembled and run on the appropriate computer*. Test data was distributed to the programmers, and a test program was defined to be debugged for the purposes of the CFA committee's work if it performed correctly on the test data.

3. S, M and R: Measures of an Architecture's Performance

Very little has been done in the past to quantify the relative (or absolute) performance of computer architectures, independent of specific implementations. Hence, like it or not, we had little choice but to define measures of architecture performance for ourselves.

Fundamentally, performance of computers is measured in units of space and time. The measures that were used by the CFA Committee to measure a computer architecture's performance on the test programs were:

Measure of Space

S: Number of bytes used to represent a test program.

Measures of Execution Time:

M: Number of bytes transferred between primary memory and the processor during the execution of the test program.

R: Number of bytes transferred among internal registers of the processor during execution of the test program.

All of the measures described in this section are measured in units of 8-bit bytes. A more fundamental unit of measure might be bits, but we faced a number of annoying problems with respect to carry propagation and field alignment that make the

* The exceptions were test programs A, B, C, and L since they all require the use of privileged instructions and it was impractical to require programmers to get stand-alone use of all the candidate machines. In these four cases, an "expert" on a test program was designated and he was responsible for reading in detail all implementations of the test program and returning the test programs to the programmer for correction if he detected any errors.

measurement of S, M, and R in bits unduly complex. Fortunately, all the computer architectures under consideration by this committee are based on 8-bit bytes (rather than 6, 7, or 9-bit bytes) and hence the byte unit of measurement can be conveniently applied to all these machines.

3.1. Test Program Size

An important indication of how well an architecture is suited for an application (test program) is the amount of memory needed to represent it. We define $S_{i,j,k}$ to be the number of 8-bit bytes of memory used by programmer i to represent test program j in the machine language of architecture k. The S measure includes all instructions, indirect addresses, and temporary work areas required by the program.

The only memory requirement not included in S is the memory needed to hold the actual data structures, or parameters, specified for use by the test programs. For example, in the Fourier transform test program S did not include the space for the actual vector of complex floating-point numbers being transformed but it did include pointers used as indices into the vector, loop counters, booleans required by the program, and save-areas to hold the original contents of registers used in the computation.

3.2. Processor Execution Rate Measures

In selecting among computer architectures, as opposed to alternative computer systems, we are faced with a fundamental dilemma: one of the most basic measures of a computer is the speed with which it can solve problems, yet a computer architecture is an abstract description of a computer that does not define the time required to perform any operation. (In fact, it is exactly this time-independence that makes the concept of a computer architecture so attractive!) Given this dilemma, one reaction might be to ignore performance when selecting among alternative computer architectures and leave it to the engineers implementing the various physical realizations to worry about execution speed. However, to adopt this attitude would invite disaster. In other words, although we were evaluating architectures, not implementations, it was essential that the architecture selected yield cost/effective implementations, i.e., the architecture must be "implementable".

The M and R measures defined below were developed to measure those aspects of a computer architecture that will most directly affect the performance of its implementations.

3.3. Processor Memory Transfers

If there is any single, scalar quantity that comes close to measuring the "power" of a computer system, it is the bandwidth between primary memory and the central processor(s) [cf. BelC71; GMLC75; StoH75].

This measure is not concerned with the internal workings of either the primary memory or the central processor; it is determined by the width of the bus between primary memory and the processor and the number of transfers per second the bus is capable of sustaining. Since processor/memory bandwidth is a good indicator of a computer's execution speed, an important measure of an architecture's effect on the execution speed of a program is the amount of information it must transfer between primary memory and the processor during the execution of the program. If one architecture must read or write $2x10^6$ bytes in primary memory in order to execute a test program and the second architecture must read or write 10^6 bytes in order to execute the same test program, then, given similar implementation constraints, we would expect the second architecture to be substantially faster than the first.

The particular measure of primary-memory/central-processor transfers used by the CFA Committee is called the M measure. $M_{i,j,k}$ is the number of 8-bit bytes that must be read or written from primary memory by the processor of computer architecture k during the execution of test program j as written by programmer i.

Clearly, there are implementation techniques used in the design of processors and memories to improve performance by attempting to reduce processor/memory traffic, i.e., cache memories, instruction lookahead (or behind) buffers, and other buffering schemes. However, with the intention of keeping our measure of processor/memory traffic as simple, clean, and implementation-independent as possible, none of these buffering techniques were considered. At the completion of one instruction, and before the initiation of the next instruction, the only information contained in the processor is the contents of the registers in the processor state.

Table 3-1 shows an example of a small IBM S/370 instruction sequence which should help to illustrate the calculation of M. The instructions are the basic loop of a routine for calculating the inner product of two single precision floating point vectors of length 10.

3.4. Registers Transfers Within the Processor

The processor/memory traffic measure just described is our principle measure of a computer architecture's execution rate performance. However, it should not be too surprising that this M measure does not capture all we might want to know about the performance potential of an architecture. In this section a second measure of architecture performance is defined: R -- register-to-register traffic within the processor. Whereas the M measure looks at the data traffic between primary memory and the central processor, R is a measure of the data traffic internal to the central processor. The fundamental goal of the M and R measures was to enable the CFA committee to construct a processor execution rate measure from M and R (ultimately an additive measure: $aM + bR$, where the coefficients a and b can be varied to model projections of relative primary memory and processor speeds). An unfortunate but unavoidable property of the R measure is that it is very sensitive to assumptions about the register and bus structure internal to the processor; in other words, the "implementation" of the processor.

The definition of R is based on the idealized internal structure for a processor shown in Figure 3-1. By using the register structure in Figure 3-1 we do not imply that this is the way processors ought to be built. On the contrary, the structure in Figure 3-1 has a much more regular data path structure than would be practical in contemporary processors. There exist both data paths of marginal utility and non-existent data paths that, if present, could significantly speed up the processor. This structure was selected because the very regular data path, ALU, and register array structure helped simplify our analysis.

$R_{i,j,k}$ is defined as the number of 8-bit bytes that are read to and written from the internal processor registers during execution of test program j on architecture k as written by programmer i.

ALU Operations. The ALU in Figure 3-1 is allowed to perform any common integer, floating point, or decimal arithmetic operation; increment or decrement; and perform arbitrary shift or rotate operations.

Only Data Traffic Measured. All data traffic is measured in R and no control traffic measured. Figure 3-1 is intended to specify what will be defined to be control traffic and what will be data traffic for the purposes of the R measure. The R measure does not count the following "control" traffic:

(1) The setting of the condition codes by the ALU (or control unit) and the use of the condition codes by the ALU. The only time that movement of data into or out of the Program Status Word will be counted in the R measure is when a Load PSW instruction is performed or a trap or interrupt sequence moves a new PSW into or out of the PSW register.

(2) Bits transmitted by the control unit to activate or otherwise control the register file, ALU, or memory unit, are not counted in the R measure.

(3) Reading of the Instruction Register by the control unit as it decodes the instruction to determine the instruction execution sequence is not counted in the R measure. In other words, the Instruction Register (with the exception of displacement fields) will be for most practical purposes a write-only register as far as the R measure is concerned.

(4) Loading the Memory Address Register is counted in the R measure, but use of the contents of the Memory Address Register to specify the address of data to be accessed in primary memory is not counted.

Virtual Address Translation. The virtual to real address translation process is not counted in the R measure. In other words, the final memory address in the MAR is a virtual address and the work involved in translating this virtual address to a real address is not included in the R measure.

The definition of the R measure was the center of considerable discussion within the CFA committee. The full set of rules that are necessary to completely define the R measure is too voluminous to present here; readers interested in the details of the R measure are referred to Volume III of the CFA Selection Committee's final report [FulS76b]. Figure 3-2 illustrates the calculation of the R measure for an IBM/370 add instruction.

4. Statistical Design of Test Program Assignments

The test program phase of the CFA evaluation process involved comparison of twelve test programs on three machines. Approximately sixteen programmers were available for the study and a complete factorial design would have required each programmer to write all of the test programs on each of the machines (for a total of 576 programs). This was clearly not feasible with the given time and resource constraints, and, consequently, a fractional design (or several fractional designs) had to be selected. Fractional factorial designs are discussed by [DavO71], e.g. The fractional designs to be described below incorporate balance in the way test program, machine, and programmer combinations are assigned.

It was necessary to consider designs which required each programmer to write test programs for all three machines. Otherwise, comparisons among the machines could not be separated from comparisons among the programmers. A desirable design would have instructed each programmer to write a total of six or nine different test programs, one third of them on each of the three machines. For most of the programmers in the study time limitations precluded this type of design, and some compromise was required. The compromise design selected also had to allow for precise comparisons among the three competing architectures. A type of design that meets both of these objectives is the nested factorial [AndV74, e.g.].

The test program part of the study actually involved the use of three separate experimental designs, henceforth referred to as Phase I, Phase II, and Phase III. Nested factorial designs were used for Phase I and Phase III. Phase II was a one-third fraction of a 3^4 factorial design. Phase I was used to study test programs A through H, those deemed to be of primary interest. Phase III was used to study test programs

I through L. Phase II included test programs A-B, E-H, and J-L. Plans of the three designs are depicted in Figure 4-1.

The Phase I design is a pair of nested factorials, each involving four programmers. Each programmer was asked to write two test programs for all three machines. Each of the eight test programs in Phase I appears once on each machine in each of the nested factorials. When this design was originally formulated, the plan included requiring programmers to write their six test programs in a preassigned randomly selected order, so as to eliminate possible biases due to learning during the course of completing the assignments. This procedure was discarded, however, when the programmers objected because of the varying availability of the three machines for debugging. Programmers were instructed to complete the assigned jobs in conformity with their typical practices and working habits with regard to order, consultation with other individuals, and other such considerations. Programmers in the study were not permitted to consult with each other, however, on any substantive matters concerning their designated assignments. All programmers were instructed to keep diaries of their work on the experiment.

As noted above, the Phase I design was formulated with the goal of obtaining maximum possible information about differences between the competing architectures. With the given Phase I design, comparisons among the three architectures are not confounded by effects of either test programs or programmers. The Phase I design called for 48 observations and was viewed as the most important of the three designs formulated.

The design termed Phase III was formulated according to the same plan as was Phase I, except that four test programs and four programmers were utilized. The

Phase III design contains half as many observations as the Phase I design and thus gives statistical results of less precision. The test programs in the Phase III design are of lesser interest than those in Phase I. The four programmers in Phase III are distinct from the eight in Phase I.

Together the Phase I and Phase III designs provide a view of all three machines and the operation of all twelve test programs selected for consideration. A third experiment, labelled Phase II, was also planned . This was viewed as an auxiliary effort and was to be completed only if it was clear that the programmers assigned to it would not be needed to aid in the completion of Phase I and Phase III. The Phase II design called for three programmers to write nine different test programs, three on each of the three machines. The programmers assigned to Phase II were able to devote enough time to the test program study to permit use of a design which required them to write nine different programs. Some comparisons among programs not possible in Phase I and Phase III could be made, and the statistical results of Phase II could be compared to those of the other two experiments. The design used was the 3.4.3 plan in [ConW59]. This was made possible by dividing the factor representing test programs, which appears at nine levels, into two pseudofactors (see [AndV74]), each at three levels. One of the Phase II programmers also participated in the Phase I design. The only duplicate assignment, however, was test program G on the IBM S/370.

5. Analysis of Test Program Results

This section describes the experimental results and statistical analysis of the test program data. We shall first discuss the Phase I experiment, then the Phase III experiment, and then the analysis combining data from Phase I and III. Finally, the Phase II experiment will be described.

5.1. Phase I Models

A possible model for the nested factorial designs in Phase I is

$$Y_{ijk} = C + P_i + T_{ij} + M_k + PM_{ik} + TM_{ijk} + e_{ijk} \qquad (5.1)$$

$i = 1,2,3,4$ $j = 1,2$ $k = 1,2,3$

In this equation y_{ijk} is some response (i.e., on S, M, or R measure) generated by the ith programmer writing the jth test program on the kth machine. Also,

C	= constant, termed the grand mean
P_i	= effect due to the ith programmer
T_{ij}	= effect of the jth test program assigned to the ith programmer
M_k	= effect of the kth machine
PM_{ik}	= interaction between the ith programmer and the kth machine
TM_{ijk}	= interaction between the jth test program written by the ith programmmer and the kth machine
e_{ijk}	= a random error term, assumed to be normally distributed with mean 0 and variance not dependent ont the values of i, j, and k.

The Phase I experiment may also be modelled in a manner somewhat different from that just described. In Phase I there are two factors at eight levels each, programmers and test programs, and one factor at three levels, machines. The two eight-level factors may each be replaced by three pseudofactors at two levels each. Then we are concerned with a complete factorial experiment involving $3*2^6 = 192$ total observations. The actual Phase I experiment is a 1/4 fraction of this. A model may be fit using dummy variables to account for various effects and interactions.

5.2. Transformation of the Data

Examination of the S, M, and R data values collected clearly shows there is wide variation in the data from one test program to another, e.g., especially for the M and R measures. Various statistical considerations suggest that some transformation of the raw data prior to analysis is desirable. A technical discussion of transformation of statistics is given by [RaoC73], who illustrates use of the methodology in various contexts.

In the CFA study the purpose of a transformation of the data is to stabilize variance, so that an additive model such as (5.1) will hold for each of the designs. Specifically, the model (5.1) assumes that the variance of the error term e_{ijk} is independent of i, j, and k. Under this assumption inferences which follow from analysis of variance (ANOVA) calculations, as described below, are valid.

A variance stabilizing transformation is frequently suggested by consideration of the experimental situation and prior understanding of the variation to be expected in the data. For example, consider the M and R measures. Suppose some programmers each write two test programs and the average run time of the second one is k times the average run time of the first. Then if the standard deviation of the M or R readings is V for the first test program, it can be expected to be proportional to kV for the second test program. In other words, the variability (standard deviation) in run times is directly proportional to the average run time. The accuracy of this conjecture may be tested by examination of the data, but clearly there is strong intuitive support for it. Consider the Runge-Kutta test program. Its M and R measures are dominated by the computation of the inner loop performing the step-wise solution of the differential equation. Variations in M and R measures will be a result of alternative encodings of this inner loop. Average M and R measures will be doubled if the number of iterations requested is doubled. Moreover, doubling the number of iterations will also cause the differences between the different Runge- Kutta programs to double. When the standard deviation of the test data is directly proportional to the mean, a logarithmic transformation will stabilize the variance, that is, remove the dependence of the variance on the size of the test program [RaoC73, Section 6g.1].

The model of (5.1) may be termed an additive model. When a logarithmic transformation is used for the data, y_{ijk} in (5.1) becomes the logarithm of the response, such as the M or the R reading. In this case a multiplicative model in fact underlies (5.1) and we write

$$z_{ijk} = \chi \pi_i \tau_{ij} \mu_k \pi\mu_{ik} \tau\mu_{ijk} \epsilon_{ijk}, \qquad i = 1,2,3,4, \quad j = 1,2, \quad k = 1,2,3. \tag{5.2}$$

The connection between (5.1) and (5.2) is

$$\begin{aligned}
\ln z_{ijk} &= y_{ijk}, \\
\ln \chi &= c, \\
\ln \pi_i &= P_i, \\
\ln \tau_{ij} &= T_{ij}, \\
\ln \mu_k &= M_k, \\
\ln \pi\mu_{ik} &= PM_{ik}, \\
\ln \tau\mu_{ijk} &= TM_{ijk}, \\
\ln \epsilon_{ijk} &= e_{ijk}.
\end{aligned}$$

Thus, use of the logarithmic transformation on both sides of (5.2) yields (5.1), and the multiplicative model (5.2) may be viewed as the meaningful basic underlying model.

Similarly, consideration of the underlying properties of the S measure suggest a square root transformation is appropriate to stabilize its variance. This transformation arises because the variance, rather than the standard deviation, of the S measure can be expected to be proportional to kV (See [FulS76b]). Use of the square root transformation would imply use of the model in (5.1) with y_{ijk} denoting the square root of the measured S value.

It should be noted that the square root and logarithmic transformations are only two of a large number of possible transformations. A particular family of transformations takes a response z and transforms it according to z^a for an $a > 0$. With an appropriate interpretation, the logarithmic transformation corresponds to the limiting value $a \rightarrow 0$. This family of power transformations is discussed in detail by [BoxG64].

5.3. Statistical Analysis of Phase I Data

ANOVA calculations were performed on both halves of the Phase I experiment for $\sqrt{S}$, $\ln M$, and $\ln R$ values. In each analysis the sample variance of the 24 values was decomposed into sums of squares attributable to variations among programmers test programs, machines, programmer- machine interactions, and test program-machine interactions. The proportions of the total variance due to the various sums of squares are given in Table 5-1 of [FulS76]. The ANOVA calculations indicate that test program and programmer variations account for most of the variation in the data in the case of the M and R measures, and that machine differences are relatively small. Machine differences are more noticeable for the S measure.

Using dummy variables, we also fit models using the formulation discussed at the end of Section 5.1. In each model 24 parameters were fit, leaving 24 degrees of freedom to measure experimental error. Estimates of the variance of the error term in the model (5.1) are 18.175, 0.377, and 0.400 for $\sqrt{S}$, $\ln M$, and $\ln R$, respectively. The actual data values for the S, M, and R measures are given in the Appendix, and these estimates of variance reflect the magnitude of the experimental error component in the model (5.1). Table 5-1 shows estimates of various machine comparisons for the Phase I data. A 95% confidence interval is quoted below each estimate. The 95% confidence intervals which do not cover the value 0 correspond to comparisons statistically significant at level 0.05(=1-.95). Thus at level .05 the Interdata 8/32 is superior to the IBM S/370 on all measures. The PDP-11 is adjudged superior to the IBM S/370 at level .05 on two of the measures and barely misses being superior when $\sqrt{S}$ is considered. Moreover, the IBM S/370 is inferior to the average performance of the other two machines on all measures. It is worth noting that these comparisons among the competing architectures are based upon consideration of test programs A through H only. It is reasonable, however, to view the eight programmers in Phase I as representative of a larger population of programmers.

Table 5-2 displays estimates of the effects M_k and μ_k for the various measures. The μ_k estimates are obtained by exponentiating the estimates of M_k and are appropriate for the logarithmic models only. Estimates have been included for architecture comparisons obtained from the model (5.1) with the response ln S. These are also given in Tables 5-4 and 5-6 below. Use of the ln S model leads to estimates which are qualitatively similar to those obtained from the $\sqrt{S}$ model, and it permits more convenient comparisons of the three architectures. Since the effects noted in Table 5-2 are differential values, a value of 0 is neutral for M_k and a value of 1 is neutral for μ_k. The figures in Table 5-2 are consistent for the different measures and transformations. The IBM S/370 is noticeably worse than the other two architectures. For all but the ln R response, the Interdata 8/32 appears to be modestly better than the PDP-11.

One may interpret the last three lines of Table 5-2 in the following way. The ln M measure results indicate the IBM S/370 requires 155.7% as many processor/memory transfers to "execute" programs A through H as the average of the three machines, while the PDP-11 and Interdata 8/32 require 79.5% and 80.9%, respectively.

5.4. Phase III Models and Results

The models for Phase III experiments are the same as in (5.1) and (5.2), except that the subscript i assumes the values 1 and 2 only. Estimates of the variance of the error term in the Phase III version of model (5.1) are based on eight degrees of freedom and are 18.606, 0.374, and 0.308 for $\sqrt{S}$, ln M, and ln R, respectively.

Table 5-3 is the analog of Table 5-1, and Table 5-4 the analog of Table 5-2. None of the confidence intervals shown in Table 5-3 fails to cover the value 0. However, it is apparent that the PDP-11 performed noticeably worse than the other two machines in Phase III. Also, there is very little difference between the IBM S/370 and the Interdata 8/32 in Phase III.

The relatively poor performance of the PDP-11 in Phase III appears to be due to its inability to handle test program I, quicksort. Certainly part of the explanation for the poor performance of the IBM S/370 in Phase I can be attributed to test program A, I/O kernel with four priority levels. In the next section results from Phase I and Phase III are combined to produce overall estimates of machine effects and overall comparisons of the machines.

5.5. Combination of Phase I and Phase III Results

Let θ_I denote an estimate of a machine effect or comparison, such as M_1 or M_3-M_1, in Phase I. Let θ_{III} denote the estimate of the same effect or comparison in Phase III. In the previous two sections such estimates were given, as well as some confidence intervals. The purpose of this section is to present estimates of the form

$$\alpha\theta_I + (1-\alpha)\theta_{III} \tag{5-3}$$

where α is chosen to minimize the variance of the resulting linear combination and $0 < \alpha < 1$. Table 5-5 shows estimates of machine comparisons and 95% confidence intervals. The value of α for each column in the table is given along the top border. In all columns more weight is given to the Phase I data. Table 5-6 gives estimates of machine effects with Phase I and Phase III data combined.

All of the confidence intervals for M_3-M_2 in Table 5-5 fail to cover the value zero. Thus, the evidence suggests that the Interdata 8/32 performs better than the IBM S/370 on all three measures, S, M, and R. Also, the IBM S/370 tends to be worse than the average of the other two machines.

The estimates of μ_k in Table 5-6 provide a summary of the Phase I and Phase III data. The IBM S/370 requires 120.8% as much storage as the average of all three machines for the twelve test programs studied. According to the ln M measure estimate, the IBM S/370 required 126.6% as many processor/memory transfers to "execute" the test programs as the average of the three machines. The other figures in the lower part of Table 5-6 are interpreted similarly.

5.6. Phase II Models and Results

Analysis of variance calculations were performed on data arising from the Phase II design. Some of the results for responses $\sqrt{S}$, ln R, and ln M are summarized in Table 5-7. This table indicates the proportions of the total variance attributable to various sums of squares. The variance was split into sums of squares each with two degrees of freedom. Since two of the factors in the design were in fact pseudofactors at three levels each to account for the nine test programs, several sets of sums of squares were combined. There is some aliasing in the design involving the second-order interactions.

Evaluation via Test Programs

Estimates of differential effects in a model comparable to (5.1) for the three machines can also be given. For the $\sqrt{S}$ measure they are -.952 for the PDP-11, 1.605 for the IBM S/370, and -.653 for the Interdata 8/32. For the ln M measure the values are -0.691, 0.508, and 0.183 for the machines quoted in the same order, and the figures are -.662, .538, and .123 for the ln R measure. Thus, the experimental results for this phase tend to rank the machines with the PDP-11 first by a substantial margin, and the Interdata 8/32 ranks second. However, it should be noted that test program A was included in the Phase II design, and test programs D and I were not.

6. Summary

This article has described how the test program phase of the CFA study was developed, what methodologies were used, and what were the results of the study. We began with a discussion of the twelve test programs used in this study and how the CFA committee selected these twelve from a larger set of test programs as most representative of the expected applications of military computers. A Program Definition Language (PDL) was used to clearly specify these test programs so that it was clear to the programmers exactly what algorithm was to be implemented yet also indicate to what extent we expected the programmer to optimize the coding of the test programs to take advantage of the features of the architecture under test.

Section 3 of this article defined the three measures of performance used to evaluate the candidate computer architectures on each test program:

S: The number of bytes used to represent a test program

M: The Number of bytes transfered between primary memory and the processor during execution of the test program

R: The number of bytes transfered among internal registers of the processor during execution of the test program

Evaluation via Test Programs

The test programs were assigned to programmers based on a statistical design involving three phases, denoted as I, II, and III. In Phase I eight programmers were assigned two test programs to implement on each of the three machines. Phase III was a smaller version of Phase I, involving only four programmers. Phase II was a somewhat more complex design that involved each of three programmers writting nine different test programs, three on each machine. Phase II was intended to give some information on the interaction between particular test programs and machines that was not available with much precision from Phases I and III.

The principal results of the test program study that were passed along to the life-cycle cost models [CorJ77] was the composite performance of the candidate architectures for phases I and III on the set of 12 test programs. An analysis of Variance (ANOVA) procedure was used to determine the overall relative performance of the three candidate machines, as shown in Table 6-1. Unity indicates average performance and the lower the score on any of the measures, the better the machine handled the set of test programs.

In other words, the test program results indicate that the IBM S/370 needs 46% more memory than the Interdata 8/32 to represent the set of test programs (or 21% more than the average of the three architectures) and the PDP-11 is essentially average in its use of memory.

Considering the test program results in a little more detail, in Phase I the data revealed the IBM S/370 to be significantly worse than the other two machines on S, M, and R measures at a significance level of 0.05 (i.e. the 95% confidence intervals all failed to include the point where the IBM S/370 equals the performance of the other machines). Moreover, the overall performance of the PDP-11 was virtually identical to that of the Interdata 8/32. Some part of the poor performance of the IBM S/370 can be traced to test program A (the priority I/O kernel). In Phase III alone, none of the comparisons among the three machines was significant at the 0.05 level because of the small number of data points (24). However, the PDP-11 was noticeably the worst of the three machines on all three measures. The IBM S/370 dominated the Interdata 8/32 with regard to the M measure, the Interdata was better for the S measure, and there was little difference between the two for the R measure. The relatively poor performance of the PDP-11 appeared to be due to the quicksort test program, test program I, which worked with a list much larger than the 64k byte virtual address space of the PDP-11.

Statistical results from Phases I and III were combined. In this analysis the ranking of the three machines from best to worst on the three measures was: Interdata 8/32, PDP-11, and IBM S/370. The average performance of the three architectures in Phases I and III is given in Table 6-1.

The outcome of Phase II largely corroborates the results of the other two experiments. The ranking of the three machines, from best to worst is: PDP-11, Interdata 8/32, IBM S/370. This ranking prevails for all three measures, S, M, and R. It is important to recall (See Table 4-1) that Phase II included test program A, for which the IBM S/370 performs relatively poorly, and does not include test programs D and I, which are relatively difficult to implement on the PDP-11, because they have large data structures. Because of the magnitude of the experimental error in these test programs and the relatively small number of data points in Phase II (27), we were not able to detect any test program/architecture interactions that were statistically significant.

Acknowledgements

Evaluation via Test Programs

During the specification of the test programs and development of the S, M, and R measures, we had helpful discussions with many individuals related to the CFA project. Mario Barbacci, Lynn DeNoia, Robert Gordon, David Parnas, John Shore, Daniel Siewiorek, and William Smith. We are especially indebted to a group of graduate students at Carnegie-Mellon University who proved crucial to the successful completion of the full set of test programs. Three of these students, Navindra Jain, George Mathew, and Leland Szewerenko were particularly helpful through their continued effort on behalf of this project.

7. Appendix A - S, M, and R Measures for Each Test Program

On the following pages are actual measurements for each of the test programs written for the CFA program. The unit of measurement for all data is (8-bit) bytes. The number in brackets following each measurement is the identifying number of the programmer who wrote and debugged the particular test program. Data followed by an "A" are auxillary data points. Data followed by a "*" were associated with programming assignments not completed in time to be used by the CFA Committee and the pseudo-values shown were used in the ANOVA calculation (when the actual data points became available at a latter date, insertion of the real values for these programs had no significant effect on the results).

INDIVIDUAL S MEASURES

Test Program	Computer Architecture		
	IBM S/370	PDP-11	Interdata 8/32
A. Priority I/O Kernel	216 [3]	48 [4]	26 [12]
	286 [12]	32 [12]	28 [14]
	742 [14]	32 [14]	26 [17]
B. FIFO I/O Kernel	372 [2]	133 [2]	144 [2]
	465 [13]	124 [3]	142 [4]
	308 [17]	246 [13]	98 [13]
C. I/O Device Handler	192 [1]	132 [1]	176 [1]
	252 [17]	216 [17]	241 [17]
D. Large FFT	454 [11]	766 [11]	550 [11]
	454 [9]*	766 [9]*	402 [9]
			402 [17]A
E. Character Search	104 [1]	88 [1]	120 [1]
	92 [4]	136 [11]	144 [3]
	154 [11]	90 [17]	168 [11]
F. Bit Test, Set, Reset	144 [9]	68 [3]	82 [4]
	122 [12]	78 [9]	90 [9]
	116 [17]	86 [12]	98 [11]A
			98 [12]
G. Runge-Kutta Int.	202 [2]	184 [2]	166 [12]
	238 [17]	172 [3]	158 [4]
		248 [17]	232 [11]A
			190 [17]
H. Linked List Insertion	144 [4]	162 [13]	148 [3]
	228 [13]	182 [14]	198 [13]
	176 [14]	194 [17]	164 [14]
I. Quicksort	340 [6]	940 [6]	426 [6]
	407 [5]	1534 [5]	524 [5]
J. ASCII to Float-Pt.	256 [4]	164 [5]	206 [3]
	441 [5]	208 [7]	238 [5]
	241 [7]	172 [17]	204 [7]
K. Boolean Matrix	224 [3]	174 [4]	156 [17]
	267 [6]	232 [6]	130 [6]
	284 [8]	284 [8]	180 [8]
L. Virtual Memory Exchange	292 [3]	254 [4]	328 [17]
	382 [7]	250 [7]	310 [7]
	414 [8]	378 [8]	334 [8]

INDIVIDUAL M MEASURE

Test Program	Computer Architecture		
	IBM S/370	PDP-11	Interdata 8/32
A. Priority I/O Kernel	212 [3]	28 [4]	28 [12]
	354 [12]	24 [12]	32 [14]
	522 [14]	24 [14]	28 [17]
B. FIFO I/O Kernel	424 [2]	208 [2]	192 [2]
	920 [13]	188 [3]	226 [4]
	434 [17]	296 [13]	114 [13]
C. I/O Device Handler	328 [1]	309 [1]	426 [1]
	304 [17]	290 [17]	279 [17]
D. Large FFT	10810 [11]	14746 [11]	10886 [11]
	10810 [9]*	14746 [9]*	8560 [9]*
			8560 [17]A
E. Character Search	854 [1]	730 [1]	958 [1]
	940 [4]	770 [11]	1044 [3]
	1724 [11]	520 [17]	1021 [11]
F. Bit Test, Set, Reset	378 [9]	162 [3]	222 [4]
	358 [12]	178 [9]	176 [9]
	238 [17]	152 [12]	296 [11]A
			276 [12]
G. Runge-Kutta Int.	141074 [2]	102662 [2]	100062 [2]
	228056 [17]	94960 [3]	100042 [4]
		176960 [17]	117984 [11]A
			138414 [17]
H. Linked List Insertion	228 [4]	204 [13]	224 [3]
	304 [13]	218 [14]	260 [13]
	264 [14]	240 [17]	238 [14]
I. Quicksort	1024 [5]	14960 [5]	2968 [5]
	1008 [6]	2756 [6]	1732 [6]
J. ASCII to Float-Pt.	241 [4]	292 [5]	363 [3]
	437 [5]	275 [7]	423 [5]
	433 [7]	283 [17]	334 [7]
K. Boolean Matrix	832 [3]	582 [4]	384 [6]
	909 [6]	776 [6]	566 [8]
	896 [8]	932 [8]	640 [17]
L. Virtual Memory Exchange	532 [3]	541 [4]	721 [7]
	532 [7]	566 [7]	1058 [8]
	645 [8]	945 [8]	780 [17]

INDIVIDUAL R MEASURES

Test Program		Computer Architecture	
	IBM S/370	PDP-11	Interdata 8/32
A. Priority I/O Kernel	947 [3]	108 [4]	166 [12]
	2146 [12]	106 [12]	166 [17]
	3052 [14]	106 [14]	214 [14]
B. FIFO I/O Kernel	2222 [2]	1096 [2]	698 [2]
	4583 [13]	810 [3]	937 [4]
	2226 [17]	1419 [13]	482 [13]
C. I/O Device Handler	1789 [1]	1480 [1]	1902 [1]
	1729 [17]	1416 [17]	1391 [17]
D. Large FFT	62904 [11]	70512 [11]	60446 [11]
	62904 [9]*	70512 [9]*	50045 [9]*
			50045 [17]A
E. Character Search	5603 [1]	4348 [1]	5885 [1]
	5549 [4]	4326 [11]	3139 [3]
	10239 [11]	3091 [17]	5767 [11]
F. Bit Test, Set, Reset	1674 [9]	832 [3]	891 [4]
	1542 [12]	917 [9]	887 [9]
	1212 [17]	801 [12]	1167 [12]
			1281 [11]A
G. Runge-Kutta Int.	845966 [2]	724372 [2]	696085 [2]
	1203952 [17]	665529 [3]	696049 [4]
		1012727 [17]	777846 [11]A
			874923 [17]
H. Linked List Insertion	950 [4]	1025 [13]	834 [3]
	1741 [13]	1087 [14]	1049 [13]
	1137 [14]	1210 [17]	965 [14]
I. Quicksort	7618 [5]	74278 [5]	13315 [5]
	7540 [6]	15205 [6]	9609 [6]
J. ASCII to Float-Pt.	1330 [4]	1726 [5]	2100 [3]
	2578 [5]	1512 [7]	2270 [5]
	2226 [7]	1716 [17]	1897 [17]
K. Boolean Matrix	5576 [3]	3180 [4]	2216 [6]
	5661 [6]	3905 [6]	3154 [8]
	5277 [8]	4446 [8]	3945 [17]
L. Virtual Memory Exchange	1931 [3]	2616 [4]	2539 [7]
	1934 [7]	2911 [7]	4573 [8]
	2529 [8]	4226 [8]	2643 [17]

```
                                 R      Comments

(1)        LA     2,10(0,0)      4      Set R2 to 10, the length of the vectors.
(2)        LA     3,XVEC         4      Load R3 with starting address of X vector.
(3)        LA     4,YVEC         4      Load R2 with starting address of Y vector.
(4)        SDR    2,2            2      Clear floating point reg. 2.
                                        Use it to accumulate inner product.
(5)        SR     7,7            2      Clear R7
                                        Use it as index into floating point vectors.

(6) LOOP   LE     4,0(7,3)       8      Load X(i) into floating point register 4.
(7)        ME     4,0(7,4)       8      Multiply X(i) by Y(i).
(8)        ADR    2,4            2      Sum := Sum + X(i) * Y(i).
(9)        LA     7,4(0,7)       4      Increment index by 4 bytes.
(10)       BCT    2,LOOP         4      Decrement loop count and branch back if not done
                                 -----
                                 26     (Loop Total)
                                 260    (Loop (6-10)* 10)
(11)       STO    2,SUM          12     Store double precision result in SUM.
                                 ----
                                 288    Grand Total
```

Table 3-1. M Measure for IBM 370 Inner Product Example

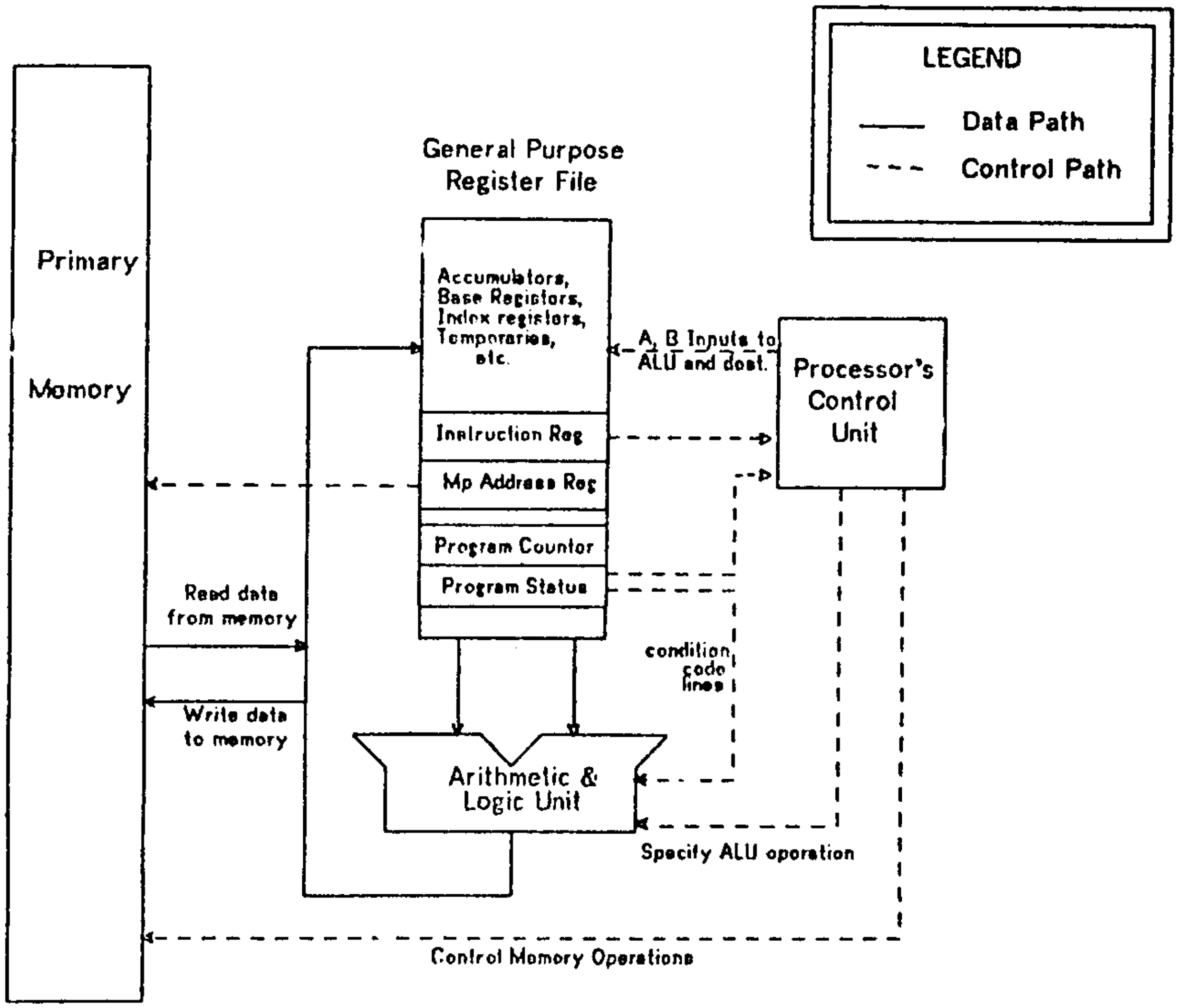

Figure 3-1: Canonical Processor Architecture

RX, RS, & SI INSTRUCTION INTERPRETATION

	R	Comment
IR<0:15> ← Mh[MAR]	2	Get halfword in instruction register
MAR ← MAR + 2	3	Incrementation counts only 1 byte
IR<15:31> ← Mh[MAR]	2	Get rest of instruction in IR
PC ← PC + 4	3	Increasing Program Counter
address interpretation	-	
instruction execution	-	
MAR ← PC	6	Set up MAR for next instruction

TOTAL	16	

RX ADDRESS CALCULATION

	R	Comment
1. B2 = 0, X2 = 0		
MAR ← IR<20:31>	5	Read 12 bits from the IR
2. B2 = 0, X2 > 0		
MAR ← IR<20:31> + R[x2]<8:31>	8	Add 12 bits from IR to 24 bits from index
3. B2 > 0, X2 = 0		
MAR ← IR<20:31> + R[B2]<8:31>	8	
4. B2 > 0, X2 > 0		
MAR ← IR<20:31> + R[B2]<8:31>	8	
MAR ← R[x2] + MAR	9	Full 24 bit (3 byte) addition

TOTAL	17	

EXAMPLE INSTRUCTION: A R4,DISP(R2,R7)

RX Add Instruction	R
RX instruction interpretation	16
address interpretation	17
MBR ← Mw[MAR]	4
R[R1] ← R[R1] + MBR	12
	--
TOTAL	49

Figure 3-2. IBM S/370 R Measure Example

Phase	Programmer	A	B	C	D	E	F	G	H	I	J	K	L
		Test Program											
I	14	all							all				
	1			all		all							
	2		all					all					
	9				all		all						
	11				all	all							
	12	all					all						
	13		all						all				
	17			all				all					
	3	370	11			832	11	11	832		832	370	370
II	4	11	832			370	832	832	370		370	11	11
	17	832	370			11	370	370	11		11	832	832
	5									all	all		
	8											all	all
III	6									all		all	
	7										all		all

Figure 4-1. Layouts of Phase I, II, and III Designs
"all" designates all three machines

Comparison of Machines \ Measure	$\sqrt{S}$	ln M	ln R
$M_3 - M_1$	-.586	.018	.012
	(-3.696,2.524)	(-.430,.466)	(-.449,.474)
$M_3 - M_2$	-3.535	-.655	-.717
	(-6.645,-.425)	(-1.103,-.207)	(-1.178,-.255)
$M_2 - M_1$	2.949	.673	.729
	(-.161,6.059)	(.225,1.121)	(.267,1.191)
$\frac{1}{2}(M_1+M_3)-M_2$	-3.242	-.664	-.723
	(-5.936,-.548)	(-1.052,-.276)	(-1.122,-.323)

	M_1:	effect of PDP-11
model (5.1):	M_2:	effect of IBM S/370
	M_3:	effect of Interdata 8/32

Table 5-1. Estimates of Machine Comparisons and 95% Confidence Intervals, Phase I

Measure	$\sqrt{S}$	ln S	ln M	ln R
Machine Effects				
M_1	-.788	-.148	-.230	-.247
M_2	2.161	.354	.443	.482
M_3	-1.374	-.205	-.212	-.235
μ_1		.862	.795	.781
μ_2		1.425	1.557	1.619
μ_3		.815	.809	.791

M_1, μ_1:	effects for PDP-11
M_2, μ_2:	effects for IBM S/370
M_3, μ_3:	effects for Interdata 8/32

Table 5-2. Estimates of Machine Effects in Models (5.1) and (5.2), Phase I

Measure / Comparison of Machines	$\sqrt{S}$	ln M	ln R
M_3-M_1	-3.806	-.295	-.348
	(-8.780,1.168)	(-1.000,.410)	(-.988,.291)
M_3-M_2	-1.585	.099	-.027
	(-6.559,3.389)	(-.606,.804)	(-.666,.613)
M_2-M_1	-2.221	-.394	-.321
	(-7.195,2.753)	(-1.099,.311)	(-.960,.318)
$\frac{1}{2}(M_1+M_3)-M_2$	.318	.247	.147
	(-3.990,4.626)	(-.364,.858)	(-.407,.701)

M_1: effect of PDP-11

M_2: effect of IBM S/370

M_3: effect of Interdata 8/32

Table 5-3. Estimates of Machine Comparisons and 95% Confidence Intervals, Phase III

Measure	$\sqrt{S}$	ln S	ln M	ln R
Machine Effects				
M_1	2.009	.133	.229	.223
M_2	-.212	.042	-.165	-.098
M_3	-1.797	-.174	-.066	-.125
μ_1		1.142	1.257	1.250
μ_2		1.043	.848	.907
μ_3		.840	.936	.882

M_1, μ_1:	effects for PDP-11
M_2, μ_2:	effects for IBM S/370
M_3, μ_3:	effects for Interdata 8/32

Table 5-4. Estimates of Machine Effects in Models (5.1) and (5.2), Phase III

Comparison of Machines \ Measure	$\sqrt{S}$ $\alpha = .67$	ln M $\alpha = .66$	ln R $\alpha = .61$
$M_3 - M_1$	-1.649	-.088	-.128
	(-4.119,.821)	(-.442,.266)	(-.517,.261)
$M_3 - M_2$	-2.892	-.399	-.448
	(-5.362,-.422)	(-.753,-.045)	(-.837,-.059)
$M_2 - M_1$	1.243	.310	.320
	(-1.227,3.713)	(-.044,.664)	(-.069,.708)
$\frac{1}{2}(M_1+M_3)-M_2$	-2.067	-.354	-.384
	(-4.207,.073)	(-.661,-.047)	(-.721,-.047)

M_1: effect of PDP-11

M_2: effect of IBM S/370

M_3: effect of Interdata 8/32

Table 5-5. Estimates of Machine Comparisons and 95% Confidence Intervals, Phase I and Phase III Data Combined

Measure	$\sqrt{S}$	ln S	ln M	ln R
Machine Effects	α = .67	α = .47	α = .66	α = .61
M_1	.135	.001	.075	.064
M_2	1.378	.189	.236	.256
M_3	-1.514	-.189	-.163	-.192
μ_1		1.001	.928	.938
μ_2		1.208	1.266	1.292
μ_3		.828	.850	.825

M_1, μ_1: effects for PDP-11

M_2, μ_2: effects for IBM S/370

M_3, μ_3: effects for Interdata 8/32

Table 5- 6. Estimates of Machine Effects in Models (5.1) and (5.2), Phase I and Phase III Data Combined

Measure		$\sqrt{S}$	ln M	ln R
Sum of Squares	Degrees of freedom			
Programmers	2	.027	.018	.026
Test Programs	8	.623	.653	.660
Machines	2	.132	.076	.068
Programmers X Machines	2	.039	.053	.047
Test Programs X Machines	8	.132	.124	.121
Test Programs X Programmers	4	.047	.076	.078

Table 5-7. Phase II ANOVA Calculations
Proportion of Variance Attributable to Each Sum of Squares

ARCHITECTURE	*S*	*M*	*R*
PDP-11	1.00	0.93	0.94
IBM S/370	1.21	1.27	1.29
Interdata 8/32	0.83	0.85	0.83

Table 6-1 Average Performance of the Architectures on the 12 test Programs.

AN ARCHITECTURAL RESEARCH FACILITY: ISP DESCRIPTIONS, SIMULATION, DATA COLLECTION

Mario R. Barbacci
Carnegie-Mellon University and
Naval Research Laboratory

Daniel P. Siewiorek
Carnegie-Mellon University and
Naval Research Laboratory

Robert Gordon
Naval Underwater Systems Center

Rosemary Howbrigg
Naval Underwater Systems Center

and

Susan Zuckerman
Naval Research Laboratory

TABLE OF CONTENTS

ABSTRACT

The objectives of this paper are twofold. In the first place we discuss some issues related to the formal description of computer systems and how these issues were handled in a specific project, the selection of a standard computer architecture for the Army/Navy Computer Family Architecture (CFA) project. The second purpose is to present a methodology for automatically gathering architectural data which can be used for evaluation and comparison purposes. We will not discuss the rationale behind the selection of specific test programs and the statistical experiment set up to ascertain the influence of the programmers, the test programs, and the machine architecture on the results. These issues belong in a companion paper.

1. Introduction

There have been many attempts to specify computer architectures in some formal notation. The CFA project included, to our knowledge, the first attempt to describe the complete instruction set of several large, commercially available architectures. The candidate architectures were the IBM S/370, DEC PDP-11, and the Interdata 8/32. The experiment described in this paper involved the preparation of formal computer descriptions, the execution of machine language programs under an instrumented simulator, and the collection of data used to evaluate the architectures. Three aspects of the experiment are important to observe: 1) We did not implement specific simulators, tailored for each architecture; the system used in this project is a general purpose computer simulator driven by a formal machine description, 2) We executed a large number of test programs *, each ranging from less than a dozen

* A total of 114 simulation runs were executed. They correspond to a total of 70 different programs (some of which called for several test cases, in other instances a test case had to be divided into separated sub-cases.) The 70 programs were divided as follows: 26 for the PDP-11, 22 for each of the IBM S/370 and Interdata 8/32.

instructions to several hundred instructions, 3) We used real programs that had been executed on actual physical machines and then used to initialize the simulators.

The Naval Research Laboratory selected ISP [BelC71] as the notation to formally describe the candidate machines. This decision was based on the availability of expertise and software support at CMU and in the fact that ISP was better suited than other candidate notations for describing a computer architecture, independently of timing and other implementation issues * . This however, does not imply that ISP is free of blemishes. Some of its virtues and defects are discussed in [BarM75]. In this paper we will point out some characteristics of the notation that prevent a complete separation between architectural and implementation details.

Volume IV of the final report of the CFA committee [BarM76b] includes the ISP descriptions of the three candidate architectures and more information about the writing and debugging of ISP descriptions. It also discusses the issue of the correctness of the ISP descriptions and other matters which could not be covered in a short paper.

Section 2 presents a brief introduction to ISP through a simplified version of the IBM S/370 ISP description. Section 3 discusses the separation of architecture vs. implementation details. Section 4 describes the Architectural Research Facility. Section 5 describes the collection of architectural data from the simulation of ISP descriptions. Section 6 concludes the paper by outlining the areas in which future work could benefit from the use of the Architecture Research Facility.

* The CFA selection committee adopted the definition of architecture proposed by the designers of the IBM S/360: "The term <u>architecture</u> is used here to describe the attributes of a system as seen by the programmer, i.e., the conceptual structure and functional behavior, as distinct from the organization of the data flow and control, the logical design, and the physical implementation"[AmdG64].

2. A Typical ISP Description

The ISP notation was developed to formalize the information normally given in basic machine manuals and to supplement or, if possible, eventually replace the "programming reference manuals". Hence its essential requirements were readability, completeness, flexibility, and brevity.

The original notation was introduced for descriptive purposes and, in the context of a book [BelC71], certain ambigueties were permitted. For more formal uses, the notation had to be revised and a language named ISPL was developed between 1973-1975 [BarM76a]. Further developments on the notation continue at CMU, and a language tentatively named ISPS is being implemented. For the remainder of this paper we shall refer exclusively to ISPL, the dialect used in the description of the CFA architectures.

The example shown in Figure 1 is derived from the IBM S/370 ISP description. We will only present the main declarations and the instruction interpretation cycle *.

The control flow for all instructions in Figure 1 follows a well defined path. The main body of the ISP description is defined by the Run procedure which continuously performs a loop of instruction cycles (IFetch followed by IExec). After an instruction has been executed, a special section of code (INT) is executed. INT checks for the presence of exceptional conditions (errors or external interrupts) and performs the proper context switching to handle these conditions.

The instruction fetch section (IFetch) reads the first half-word of the instructions and from the first two bits (Instr<0> and Instr<1>) it computes the length of the

* In order to keep the examples within the space limitations of this paper, we have taken some minor liberties with the syntax of ISPL. These alterations should not overly confuse readers familiar with ISPL.

instruction (PSW<32:33>) and updates the program counter (PSW<40:63>). IFetch then proceeds to read one or two more half-words, the rest of the instruction.

The instruction execution section (IExec) uses the first two bits of the instruction (Instr<0:1>) to select an instruction-type specific section. The RR, RX, RSSI, and SS sections handle the corresponding instruction types. RX, RSSI, and SS begin by computing the effective address of the operand(s). After this step is completed the next 6 bits of the instruction (Instr<2:7>) are used to select a "routine" which describes the behavior of the instruction.

If any errors are detected during the instruction cycle (address boundary errors, illegal operations, storage protections, etc) the rest of the instruction is aborted and the proper error code is set in the PSW. This premature termination allows the interrupt handler (INT) to take care of the situation (the usual mechanism is to switch PSWs thus automatically starting the execution of interrupt specific system routines).

We have tried to keep the example as simple as possible by avoiding any details beyond those extrictly necessary to follow the example. In particular, the reader might have noticed that we were making explicit references to fields of the Instruction Register (Instr) and the Program Status Word (PSW). It is clear that when we deal with large descriptions such explicit references tend to become cumbersome and error prone *. The following section deals with the issues of how to improve the readability and writeability of ISP descriptions by using abstractions like pseudo-registers, procedures, temporary registers, etc.

* Even though some portions of the Architectures were left out of the ISP descriptions, notably the Floating-Point Instructions, the ISP descriptions used in this project are non-trivial computer programs. Each description takes between 30 and 40 pages of code. The size of the descriptions (1445 lines for the PDP-11, 2345 lines for the Interdata 8/32, and 2132 lines for the IBM S/370) reflects the size of the instruction set, not necessarily the complexity of the architecture.

3. Abstractions and Implementation Dependencies

ISP can be viewed as a programming language for a specific class of algorithms, i.e. Instruction Set Processors or Architectures. Ideally, a language to describe architectures should avoid the specification of any implementation details. Any components introduced beyond these are unnecessary for the programmer of the machine and might even bias the implementor working from the description. While these items must appear in a description of an implementation, the problem arises when describing a family of machines where the abstractions and/or algorithms may vary across members of the family. The rest of this section illustrates this problem.

3.1. Abstractions

An ISP description written using only the architectural components would not only be unreadable but also unwritable. Some form of abstraction is required. The following subsections demonstrate this point by introducing pseudo-registers, procedures, and temporary registers. These abstractions may or may not have a counterpart in some or all physical implementations of the ISP description.

Pseudo-Registers.- When writing an ISP description for a real machine it immediately becomes apparent that describing everything in terms of just the components of the architecture would lead to a cumbersome and unreadable description. The concept of a pseudo-register to rename a frequently used field of a register greatly relieves this problem. For example, consider the PDP-11 which has an autoincrement addressing mode. During the address computation an architecture register, pointed to by a subfield of the current instruction, must be incremented. Dealing only with components of the architecture would yield an expression like: R[M[Pc]<2:0>] ← R[M[Pc]<2:0>] + 2 where M[Pc] represents the current instruction in memory, pointed to by the program counter. Introducing the pseudo-register Ir (instruction register) for the current instruction would yield: R[Ir<2:0>] ← R[Ir<2:0>] + 2. We could further define a pseudo-register, Dr (for destination register), for the frequently used three bit subfield Ir<2:0>, as in: R[Dr] ← R[Dr] + 2

The pseudo-registers may suggest a register (e.g.: Ir) or a set of wires (e.g.: Dr) in some physical implementation. In reality they may have no physical correspondence at all. In any event, pseudo-registers are a useful and necessary abstraction for readable (and writable) ISP descriptions. However creating pseudo-registers for infrequently used fields or using obscure names may defeat the usefulness of this abstraction leading to reader confusion and excessive page flipping to find definitions.

Procedures.- Just as there are frequently used register fields in a machine description, there are frequently used sequences of operations. Forming these operations into procedures greatly enhances readability.

For example, consider operand fetching. Every machine has a more or less complicated effective address calculation that is performed when accessing these operands. A memory reference to a destination operand might appear as: M[Dest] where Dest is a procedure for calculating the effective address of the destination operand. Without procedures the same reference for the PDP-11 would appear as shown in Figure 2. The situation would further be aggravated if the effective address had to be processed by some form of memory management which provides for address translation and rights checking. These operations would have to be performed in the description on top of the effective address calculation. It should be noted that many minicomputers and all larger computers have some form of memory management.

Temporaries.- Occasionally readability is improved by introducing a temporary register in cases where the operands before and after the operation are required or a complex result is used repeatedly. Figure 3 shows a portion of the memory management procedures for the PDP-11.

The **Read** procedure shows the translation of a virtual address into a physical address. A temporary Memory Address Register (**Mar**) initially contains the virtual address (the result of the effective address calculation) which is then translated into a physical address in the line that reads:

```
Mar ← (PAR[Temp]<11:0> + Mar<12:6>) @ Mar<5:0> next
```

The **PAR** (Page Address Register) and **PDR** (Page Data Register) arrays contain the necessary address translation information. A bounds check is performed before the actual memory fetch from physical memory. Without the temporary variable **Mar** the **Read** procedure would be substantially complicated by having to replace every appearance of the temporary by the complex expression given above. Of course, the temporary variable may or may not have a counterpart in some implementation.

3.2. Implementation Dependencies

There are multiple examples of details that must be specified in an implementation description but do not belong in an architecture description. Typically, these are features that exhibit model dependencies. For instance, in the specification of the interrupt handling facility of a computer system, it could be the case that because of cost/performance requirements, different models must respond to simultaneous interrupts in different orders. An ISP description must by its very nature describe a specific order of interrupt trapping, thus losing a degree of freedom that one might wish to provide the machine implementors.

Figure 4 shows how the specific order in which simultaneous interrupts are fielded is build into an ISP description. Individual bits of **INTVEC** indicate the presence of a pending interrupt of a given priority. When only one interrupt is pending the proper context switching will take place. When more than one is pending there will be multiple context swaps and lower priority interrupts will be delayed to be processed later (the "new PSW" associated with a low priority interrupt will be stored into the "old PSW" position associated with a higher level interrupt).

It is not clear whether having to be specific about ordering of interrupts or similar events is a bad practice. Although one can claim that machine designers will be constrained in their choice of designs, the fact still remains that somebody must write the interrupt handling software, and for these programmers the order of interrupt fielding is important. This type of dilemma occurs quite often when dealing with ISP descriptions. The solution might be simply to write model-dependent ISP procedures whenever this conflict arises and then indicate in the ISP description which version of a given procedure must be implemented for a given model.

Another problem with implementation dependencies is that the definition of the input/output behavior of an instruction might actually imply a particular implementation. For example, consider the PDP-11 Subtract instruction. The carry condition code (C) is set according to the borrow during the subtraction. The PDP-11 Processor Handbooks describes the setting of the C bit as:

"C condition code is cleared if there was a carry from the most significant bit of the result, set otherwise."

This definition implicitly assumes that subtraction is implemented by forming the two's complement and adding. Figure 5 illustrates the situation. Consider four-bit numbers and the two methods to perform subtractions, by using a subtractor, and by using an adder after forming the two's complement.

In the adder case, the carry is the complement of the borrow which is exactly the definition given by the PDP-11 Processor Handbook. The ISP description of the setting of C becomes:

```
C ← (dest - source)<16>;                              ! Subtraction
C ← NOT (dest + NOT(source) + 1)<16>;                 ! Addition
```

As in the previous example (the order of interrupt handling), a complete algorithm had to be given. In this case, the subtractor/borrow algorithm is preferred since it presupposes only the properties of the two's complement number system. However, if an alternate implementation (such as forming the two's complement and adding) is utilized, then the implementor should be aware of possible changes in other algorithms in the ISP description.

4. The Architecture Research Facility

The facility used for the data collection phase of the CFA project is depicted in Figure 6. Reference [BarM76a] explains in full detail the features of the ISP compiler and simulator. Some familiarity with their capabilities is needed in order to understand the data collection phase described later. The following paragraphs attempt to satisfy this need.

The ISP compiler produces code for a hypothetical machine, dubbed the Register Transfer Machine (RTM). The "object code" produced by the compiler can be linked together with a program which is capable of interpreting RTM instructions. This separation between the ISP description, the RTM code, and the RTM interpreter allows the simulation of arbitrary, user defined architectures. The result of linking the RTM code with the RTM interpreter is a running program, a simulator.

The simulator accepts commands from a teletype or user designated command file. The state of the simulator can be dumped to a command file which can be read at a future date when the simulation is continued. Command files can also be used to load programs and data into the simulated target machine memory and registers.

4.1. Debugging

Most of the test programs were debugged and run on the real machines, other programs were executed exclusively under the simulator. The latter included those programs using privileged instructions that were not directly available to non-system programmers (e.g. interrupt and I/O handlers.) Results from the actual runs, whenever available, were used to check the simulated execution.

Only minor modifications and corrections were performed during the data collection phase. The largest unforeseen problem was presented by the memory management feature of the PDP-11 which was based on the PDP-11/40. The test programs which made use of this feature had been tested on a PDP-11/45 which uses different Unibus addresses for the memory management registers. This difference required minor modifications in the test programs. Most other problems were of a simpler nature and required only a few minutes to correct. It should be noted here that the simulator facility was also used to debug some programs for the Interdata 8/32 before they were executed on the real machine. This was dictated by the fact that no 8/32 was available near CMU and a large turn-around time (several days) would have complicated the debugging of the test programs.

4.2. Preparation of Simulation Tests

The ISP simulator provides commands for the loading and initialization of the simulated machine memory and internal registers. The single most important feature of the command language which permitted the fast execution and collection of statistics was the ability to read command files containing the test programs to be executed. The command language cannot handle programs in symbolic form (assembly language); it requires the preassembly of the programs into absolute, numeric, code. To get around this problem, a set of utilities was developed at CMU which permitted the transformation of assembly listings prepared by the real machine's assembler into simulation command files. This operation was performed off-line as shown in Figure 6.

Figures 7 and 8 show the transcript of a typical session using the ISP simulator. The session consists of running one of the test programs (Bit Test, Set, and Reset) on the PDP-11. The input for a simulation session consists of several files prepared off-line. These files include: The test program (derived from the assembly listing), a driver (simulation commands used to initialize the parameters for the test program), and finally, a command file with a list of those ISP procedures which must be "opaqued" (these are the procedures during which the activity counters are disabled). A typical command file, derived from an assembler listing is shown in Figure 9. This was the test program used in the sample simulator session shown in Figures 7 and 8.

4.3. Instrumentation

The ISP simulator permits the instrumentation of an ISP description by associating activity counters with each of the machine registers and memories. These counters allow the collection of statistics indicating the number of times each component of the machine is read from or written into. A separate counter is kept for each label in the ISP description. Labels are included in the ISP descriptions to identify machine instructions, addressing modes, loops (used to describe vector-like instructions like move character on the S/370), as well as other ISP procedures.

During the execution of the test programs, a data base was created by collecting dumps of the counters after each test case was completed. The files containing the counters were then processed by other, off-line, programs in order to arrive at the M and R measures.

4.4. Artificial Labels in the ISP Descriptions

Certain modifications not normally needed were made to the ISP descriptions to aid in the collection of data during the running of the test programs for the CFA project. Several labels and "do-nothing" procedures were added to identify certain phases in the instruction interpretation algorithm and to measure selected events (e.g., different addressing modes). The labels added to count these events are clearly not part of the architecture or even the implementation.

Figure 10 shows an example extracted from the S/370 ISP Description. It shows the use of artificial labels to identify different addressing modes for the RX instruction set. According to the definition of the S/360 and S/370 architectures, The RX instructions can specify both a base and an index register to be added together with the displacement field of the instruction to compute the address of the memory operand. The architecture further specifies that R[0], when specified as either a base or index register does not take place in the effective address calculation, i.e., R[0] should be specified whenever one of these two components (base or index) is missing. In the above example four dummy in-line procedures where introduced to count the number of times each possible combination of base/index modes occurs. Thus RX0000 is "executed" whenever R[0] is specified as both the base and the index register. RX00X2 is "executed" whenever R[0] is used as the base register and any of R[1:15] is used as the Index register. RXB100 is "executed" whenever R[0] is specified as the index register and any of R[1:15] is specified as the base register. Finally, **RXB1X2** is "executed" whenever R[0] is not specified as either the base or index registers. **NOP** is a dummy procedure which does not have any side effects.

5. Architecture Parameters

As a means of comparing architectures, three measures were defined for the CFA project [FulS77a]:

Measure of Space

S	The number of bytes used to represent a test program.

Measures of Execution Time

M	The number of bytes transferred between primary memory and the processor during the execution of the test programs.
R	The number of bytes transferred among internal registers of the processor during execution of the test program.

The S measure is a static parameter which can be computed independently of the ISP description. For the purposes of this paper we will restrict the discussion to the other two measures. The actual computation of the M and R measures was done through a semiautomatic process. The raw data collected from the simulator was used to count frequencies of instructions and addressing modes. These counters were multiplied by certain hand calculated factors in order to arrive at the M and R measures for each test program. Ideally, the ISP simulator should perform the entire operation and this would be a better approach, less subject to human errors. We had to use the hand computed factors due to our inability to determine the influence of the ISP writing style on the architecture parameters as defined above.

The exact methodology for writing ISP descriptions so that the M and R measures can be calculated automatically has yet to be developed. It is clear, however, that a careful control of the counting mechanism is paramount to the collection of meaningful data. During the data collection phase we made use of the following techniques towards this goal.

Opaqued Procedures.- A Simulator command allows the selective masking of in-line and off-line procedures. Masking or opaquing a procedure inhibits all activity counts inside the body of the procedure.

Certain operations, such as incrementing the program counter after an instruction, or the setting of the condition codes as a result of an instruction do not affect the R measure and should not be counted. This is typical of those actions which, in a reasonable implementation, would be done using ad-hoc circuitry, separate from the main operational units of the machine. These operations could be implemented by combinational logic (e.g.: setting condition codes from ALU lines), special registers (e.g.: using a counter instead of a simple register for the program counter), or even complex sequential networks (e.g.: the virtual address translation can be performed using its own arithmetic units and data paths).

Operations like those described above can be easily marked by adding artificial labels to the ISP description and then disabling the counters while the selected operation is being performed.

Pseudo-Register Chains.- Every component declared in an ISP description has activity counters associated with it. When a register is defined in terms of another register, such as: Pc<15:0> := R[7]<15:0>; a redefinition chain is established. Accesses higher up in the chain increment all counters lower in the chain but not vice-versa. In the above example an access of the Pc causes the register file counter for R to be incremented but accessing R[7] does not increment the program counter (Pc). By establishing appropriate redefinition chains, distinction between access types can be maintained. One variation of this technique is the use of "shadow" registers. For example two instruction registers can be defined: Ir<15:0> := Irl<15:0>; where Irl is the shadow register. The loading of the Ir from memory is to be counted in the R measure, however, the combinational logic decoding of the instruction and effective addressing mode is not to be counted. The former is performed on Ir, the latter on Irl thus distinguishing the two different types of accesses.

Memory Access Procedures.- Modern machines provide the user with an address space defined in terms of small units of information, typically 8-bit bytes. For convenience, however, the architectures also define larger access units in multiples of bytes. Thus, the IBM S/370 provides bytes, half-words, full-words, and double-words. Since the physical memory is the same, the ISP description must declare the different address spaces by building a redefinition chain in which the different address spaces are declared as "pseudo-memories" so that the M measure component of each address space is properly accounted for.

Machines like the PDP-11 add some more complexity to the issue of having multiple address spaces. The PDP-11 architecture defines the concept of an I/O page as a reserved portion of the address space, not necessarily implemented as a physical memory. Addresses in the upper 4K bytes of the PDP-11 are used to address I/O devices, machine registers, etc. Addresses in the I/O page must be handled differently when computing the M measure. If one attempts to include in-line address checks in the ISP description, the description quickly becomes bulky and unreadable. A satisfactory solution is simply to define memory access procedures (Read and Write), which can then be properly instrumented, thus enabling the automatic computation of the M measure.

<u>Temporary Registers</u>.- The automatic computation of the R measure is more difficult. In an ISP description there are three types of registers to consider: architectural, standard implementation, and temporaries. Architectural registers and certain standard implementation registers (instruction register, memory address register, and memory buffer register) can be handled using the same techniques used to automate the M measure (declaration chains and encapsulating procedures). Handling temporary registers presents a more difficult problem. The number, type, and manipulation of temporary registers are a matter of writing style.

Architecture parameters which are based solely on architecture registers while ignoring temporary registers introduced for clarity might overlook hidden computations performed on these registers. Unlike the memory, architectural registers, and standard implementation registers, a tightly defined writing style cannot be developed for temporary registers. One solution would be to use well known expression optimization techniques [WulW75] on the ISP description to uniformly minimize the temporary register activity. Hopefully the optimization would lead to similar results for equivalent algorithms.

Architectural parameters should be independent of the experience, style, and objectives of the ISP writer. This will then guarantee that the ISP descriptions which make use of abstractions (pseudo-registers, procedures, and temporary registers, etc) to enhance clarity and readability will not be penalized. By the same token, no advantage should be derived from the use of "clever" programming tricks which might attempt to bias the measurements.

6. Advantages of an Architectural Research Facility

Although for the purposes of this paper we have presented the uses of the ISPL compiler and simulator in the context of a specific project, we should point out the wider range of applications in which a system like ARF can be of great value.

6.1. A Simulator as a Training Tool

In this paper we described how machine language test programs can be executed under the simulator. The implied assumption during the data collection phase was that we were dealing with correct, finished programs. With no extra effort the ISP simulator can be a powerful training device for novice programmers. Speed of simulation is not an issue in this application. Programmers learning a new machine language tend to spend long hours single-stepping via the machine console. An interactive simulator can easily satisfy the needs of these users, while providing much better diagnostic and debugging facilities than a computer console (did you ever see a "help" button on a machine?.) ISP descriptions exist for the following machines: DEC PDP-8, PDP-10, PDP-11, IBM S/370, Interdata 8/32, and Intel 8080.

6.2. Architecture Evaluation

The S, M, and R measures are by no means the only set of architecture parameters one might wish to evaluate. Nothing in the ISP simulator depends upon this particular set of parameters. The instrumentation in the simulator allows counting every event we care to define by simply labelling the event. There is no need to create new procedures which might impact the organization or readability of the description; even a single register transfer operation can be labelled and counted.

6.3. Experimentation

Once the initial effort of writing an ISP description is accomplished, only moderate effort is required to perturb it to reflect proposed or actual changes in the architecture. Thus the effect of a modification in an architecture can be measured and studied before any funds are commited to the development of a new machine. By a careful design of the ISP description it is possible to pattern a description along the lines of the organization of the physical machine. Thus one would be able to measure and evaluate different models of the architecture. For instance, functional units and data paths can be represented by separate procedures in the ISP description. An ISP description could then be parameterized to invoke these procedures in different order, concurrently or sequentially, with or without intermediate steps, etc. as the different models differ in their implementation. An example might be determining the effect of a cache memory on the apparent instruction execution speed in high performance implementations.

6.4. Machine Relative Software

As the number of different architectures coming into existence increases every year, it is becoming more and more expensive to develop the necessary software support base that allows the effective use of these machines. The availability of user micro-programmable machines enlarges the space of possible architectures to the point that automatic software generation systems will become a necessity. Tools that operate relative to a computer description could represent a significant breakthrough in the manner that computer systems (hardware/software) are designed and evaluated. The Advanced Research Projects Agency (ARPA) of the Department of Defense is currently sponsoring this area of research at CMU and elsewhere [BarM74].

Architectural Research Facility

In the future one can foresee hardware and software automation systems that take as input computer descriptions, and language and problem specifications; and from these, generate operating systems, compilers, and other support and application software automatically. Other areas of current research include automatic diagnostic generation, microcode generation, machine verification, etc.

Formal computer descriptions will play an increasing and important role in the evaluation, procurement, verification, and programming of computers. The ARF facility is a step in this direction.

```
S370:=
begin  declare
        Memory[0:"FFFFFF]<0:7>;                                ! Primary Memory
        R[0:15]<0:31>;                                 ! General Purpose Registers
        PSW<0:63>;                                           ! Program Status Word
        . . . . . . .                   ! Auxiliary Registers (Instr, Mar, Mbr, etc.)
        eralced                                          ! End of Declarations

Run:=  begin                                             ! Main Executable Program
        IFetch:= begin                                   ! Instruction Fetch Section
                Mar←PSW<40:63> next                     ! Initial Instruction Address
                Instr<0:15>←Memory[Mar:Mar+1] next  ! Read First Half-Word of Instruction
                PSW<32:33>←Instr<0>+Instr<1>+1 next          ! Instruction Length
                PSW<40:63>←PSW<40:63>+PSW<32:33>*2 next         ! Program Counter
                . . . . . . .                     ! Fetch the rest of the Instruction
                end;
        IExec:= begin                                 ! Instruction Execution Section
                decode Instr<0:1> =>                        ! Select Instruction Type;
                RR:=    begin                         ! RR Instruction Decode Table
                        (decode Instr<2:7> => . . . . . )      ! Select RR Instructions
                        end;
                RX:=    begin                         ! RX Instruction Decode Table
                        Mar←Instr<20:31> next                          ! Displacement
                        (if Instr<16:19> => Mar←Mar+R[Instr<16:19>]) next        ! Base
                        (if Instr<12:15> => Mar←Mar+R[Instr<12:15>]) next       ! Index
                        (decode Instr<2:7> => . . . . . )      ! Select RX Instructions
                        end;
                RSSI:=  begin          ! RS,SI Instruction Decode Table
                        Mar ← Instr<20:31> next                        ! Displacement
                        (if Instr<16:19> => Mar ← Mar+R[Instr<16:19>]) next      ! Base
                        (decode Instr<2:7> => . . . . . )    ! Select RS, SI Instructions
                        end;
                SS:=    begin                         ! SS Instruction Decode Table
                        AMar1←Instr<20:31>; AMar2←Instr<36:47> next    ! Displacements
                        (if Instr<16:19> => AMar1←AMar1+R[Instr<16:19>]);        ! Base
                        (if Instr<32:35> => AMar2←AMar2+R[Instr<32:35>]) next    ! Base
                        (decode Instr<2:7> => . . . . . )      ! Select SS Instructions
                        end;
                end;
        INT:=   begin . . . . . end next                  ! Interrupt Handling Section
        Run                                                  ! Repeat Main Procedure
        end
end
```

Figure 1 - A Simplified Version of the IBM S/370 ISP Description

```
M[   decode Dd =>
          (decode Dm =>                                          ! Direct Addressing
                    #37400@Dr;                                   ! Register Mode
                    R[Dr]←R[Dr]+2 next R[Dr]-2;                  ! Autoincrement Mode
                    R[Dr]←R[Dr]-2 next R[Dr];                    ! Autodecrement Mode
                    M[Pc+2] + R[Dr]                              ! Index mode
                    );
          (decode Dm =>                                          ! Deferred Mode
                    M[#37400@Dr];                                ! Register mode
                    R[Dr]←R[Dr]+2 next M[R[Dr]-2];               ! Autoincrement Mode
                    R[Dr]←R[Dr]-2 next M[R[Dr]];                 ! Autodecrement mode
                    M[M[Pc+2] + R[Dr]]                           ! Index mode
                    )
     ]
```

Figure 2 - Inline Effective Address Calculation

```
Read:=begin
        Temp ← Mar<15:13> next
        Mar ← (PAR[Temp]<11:0> + Mar<12:6>) @ Mar<5:0> next    ! Compute Physical Address
        (if not PDR[Temp]<2:1> => Abort) next
        (if (Mar<12:6> gtr PDR[Temp]<14:8>) and not PDR[Temp]<3> => Abort) next
        (if (Mar<12:6> lss PDR[Temp]<14:8>) and PDR[Temp]<3> => Abort) next
         . . . . . . .                                         ! Read from Physical Memory
        end;
```

Figure 3 - A Portion of the PDP-11 Memory Management

```
Int:=   begin
        Temp←PSW<32:33> next                          ! Save Instruction Length
        (if INTVEC<0> AND PSW<13> =>                  ! Handle Priority (1) Interrupts
                . . . . . . .
                ) next
        (if INTVEC<1> =>                              ! Handle Priority (2) Interrrupts
                . . . . . . .
                ) next
        (if INTVEC<2> =>
                . . . . . . .
                ) next
        (if INTVEC<3> AND PSW<0:7> =>                 ! Handle Priority (3) Interrupts
                . . . . . . .
                ) next
        (if INTVEC<4> AND IOMSK =>                    ! Handle Priority (4) Interrupts
                . . . . . . .
                ) next
        PSW<16:31>←0; PSW<32:33>←Temp ! Reset Instruction Length & Interrupt Code
        end;
```

Figure 4 - Explicit Interrupt Processing Order in the IBM S/370

```
5 - 3 = 2 (no borrow)   3 - 5 = -2 (borrow)

  0101                    0011      Subtracting
  0011                    0101
  ----                    ----
0 0010                  1 1110
borrow                  borrow

  0101                    0011      Adding Two's Complement
  1101                    1011
  ----                    ----
1 0010                  0 1110
carry                   carry
```

Figure 5 - Implementation Dependant Condition Code Setting

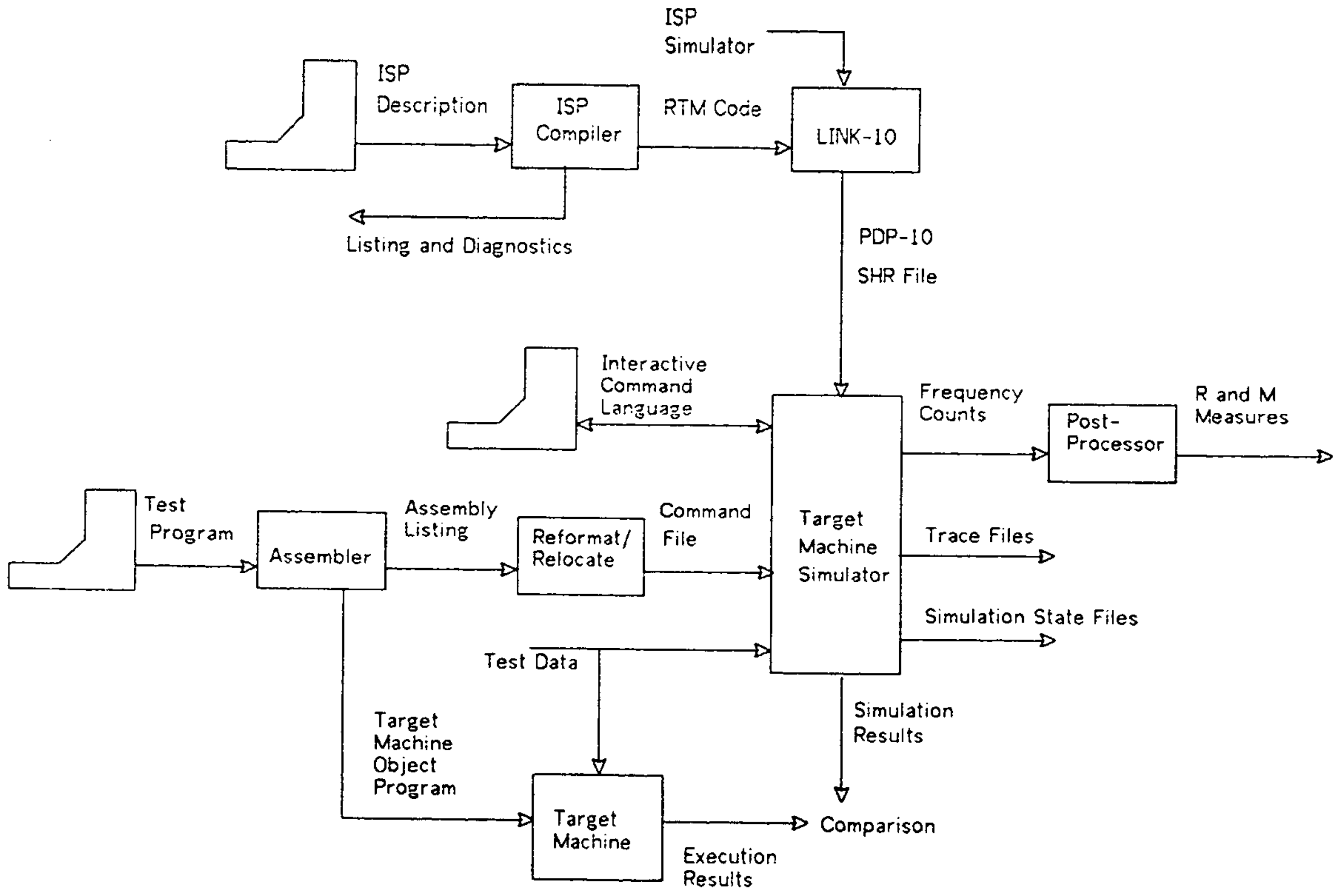

Figure 6 - Test Program Execution Under ARF

```
ru pdp11m
ISP SIMULATOR V3 - NRL ARF STAGE 2
Friday 10 Sep 76 17:13:50 PDP11M.ISP[L410MB25]
SERIALIZATION COMPLETED
SPACE ALLOCATED
TYPE HELP FOR HELP
TYPE <ESC> TO INTERRUPT SIMULATION LOOPS

>read fadi.sim                  ! Read in the benchmark file
>>RADIX OCTAL
>>DECHO                         ! The benchmark file disables the listing
                                ! on the user terminal.
>>100 LINES READ
>read fa.dr3                    ! Read in the driver file
>>!     HERE COMES THE DRIVER (CALLS)
>>SETVAL MW[3000]←013746 005202         !       MOV     @#5202,-(SP)    ; F
>>SETVAL MW[3002]←013746 005204         !       MOV     @#5204,-(SP)    ; N
>>SETVAL MW[3004]←012746 004000         !       MOV     #4000,-(SP)     ; A1
>>SETVAL MW[3006]←012746 005200         !       MOV     #5200,-(SP)     ; RC
>>SETVAL MW[3010]←012746 005206         !       MOV     #5206,-(SP)     ; W
>>SETVAL MW[3012]←004737 001000         !       JSR     PC,@#1000       ; BTSR
>>SETVAL MW[3014]←000000                !       HLT
>>      ! The above sequence of PDP-11 instructions pushes the parameters
        ! onto the stack, call the benchmark as a routine, and halt.
>>SETVAL MW[2000]←123457 071234 167006 145670   !       BIT STRING
>>SETVAL MW[2500]←0             !       RETURN CODE
>>SETVAL MW[2501]←2             !       F
>>SETVAL MW[2502]←25            !       N
>>SETVAL MW[2503]←0             !       WORK AREA
>>SETVAL PC←6000
>>SETVAL SP←200
        ! The above sequence initializes the data (parameters), the stack
        ! pointer and the program counter (which now points to the code
        ! sequence that pushes the parameters and call the routine.
>>SETVAL A←0    ! This is an ISP internal variable - indicates whether the
                ! machine is running, halted, or waiting.
>>SETCTR ALL 0,0                ! Reset activity counters
>>READ OPQ11.SIM[L410MB25]      ! PDP11 Opaqued Procedures
>>>DECHO
>>>53 LINES READ
>>READ UUO11.SIM[L410MB25]      ! UNIMPLEMENTED OPERATION BREAKS
>>>DECHO
>>>15 LINES READ
>>TRACE IR,PC,R,MWIO            ! Trace a few selected registers
                                ! IR is the Instruction Register,
                                ! PC is the Program Counter (R[7]),
                                ! R[0:7] are the general registers,
                                ! MWIO is the I/O page (R is mapped onto MWIO)
>>BREAK JSR,RTS                 ! Break on selected instructions
>>26 LINES READ
```

Figure 7 - Initialization of a Simulation Run

```
>start inter                                ! Here we start the simulation
@ INTER  + 15    IR      = 13746
@ INTER  + 20    PC      = 6002
@ SINCD  + 22    R       [ 7]= 6004
@ DDECRD + 21    R       [ 6]= 176
@ INTER  + 15    IR      = 13746

 . . . . . . . . .       ! Pushing Parameters

@ INTER  + 15    IR      = 12746
@ INTER  + 20    PC      = 6022
@ SINCD  + 22    R       [ 7]= 6024
@ DDECRD + 21    R       [ 6]= 166
@ INTER  + 15    IR      = 4737
@ INTER  + 20    PC      = 6026
BREAK AFTER JSR                             ! The simulation stops on a breakpoint
*setctr all 0,0                             ! The real benchmark starts here, we must
                                            ! reset all counters (they were modified
                                            ! during the benchmark calling sequence)
*cont                                       ! we continue the simulation
@ DINCRD + 22    R       [ 7]= 6030
@ JSR    + 14    R       [ 7]= 6030
@ JSR    + 15    PC      = 1000
@ INTER  + 15    IR      = 10046
@ INTER  + 20    PC      = 1002

 . . . . . . . . ! Program Execution Trace

@ INTER  + 20    PC      = 1072
@ SINCD  + 22    R       [ 6]= 164
@ WRITE  + 131   MWIO    [ 3740001]= 0
@ INTER  + 15    IR      = 207
@ INTER  + 20    PC      = 1074
BREAK AFTER RTS                             ! the simulation stops at the end of the
                                            ! benchmark (the return instruction)

*outctr fadl.rm3                            ! we dump all the counters into a file
*cont                                       ! we continue the simulation
@ RTS    + 2     PC      = 1074
@ RTS    + 7     R       [ 7]= 6030
@ INTER  + 15    IR      = 0
@ INTER  + 20    PC      = 6032
SIMULATION COMPLETED                        ! we executed the Halt instruction
RUN TIME(10 usec units)=831678
RTM OPS EXECUTED=4535
>exit                                       ! we finish the session
EXIT
```

Figure 8 - Program Execution Trace

```
RADIX OCTAL
DECHO
!CFAF   MACN11   V003F   5-JUL-76   12:54   PAGE 1
!BTSR1  M11
!
 . . . . . . . . ! Program, Programmer Identification (Supressed)

!     13                                    01300   ; Offsets of parameters from stack p
!     14                                    01400   ;
!     15            000004                  01500   SAVE=4          ; we need to save 2
!     16                                    01600   ;
!     17            000016                  01700   F=12+SAVE       ; function code
!     18            000014                  01800   N=10+SAVE       ; relative bit numbe
!     19            000012                  01900   A1=6+SAVE       ; address of bit str
!     20            000010                  02000   RC=4+SAVE       ; address of return
!     21            000006                  02100   WORK=2+SAVE     ; address of work ar
!     22                                    02200   ;
!     23      000000'                       02300   BTSR:
!     24      000000' 010046                02400           MOV     R0,-(SP)
!     25      000002' 010146                02500           MOV     R1,-(SP)
!     26      000004' 005076  000010        02600           CLR     @RC(SP)        ; ze
!     27      000010' 016600  000014        02700           MOV     N(SP),R0       ; ge

 . . . . . . . . ! Relocatable Object Code Listing

!     41      000066' 012601                04100   QUIT:   MOV     (SP)+,R1       ; ex
!     42      000070' 012600                04200           MOV     (SP)+,R0
!     43      000072' 000207                04300           RTS     PC
!     44      000074' 150110                04400   SET:    BISB    R1,@R0         ; FC
!     45      000076' 000773                04500           BR      QUIT
!     46              000001                04600           .END

 . . . . . . . . ! Cross-Reference Listing

!
                              ! Here begin the simulation commands
                              ! derived from the above listing
                              ! relocation address = word 400 (octal) = byte 1000
!
SETVAL MW[400]←010046
SETVAL MW[401]←010146
SETVAL MW[402]←005076 000010
SETVAL MW[404]←016600 000014

 . . . . . . . . ! Target Machine Program Loading

SETVAL MW[433]←012601
SETVAL MW[434]←012600
SETVAL MW[435]←000207
SETVAL MW[436]←150110
SETVAL MW[437]←000773

ECHO
```

Figure 9 - A Command File Derived from an Assembly Listing

```
RX:=    begin
        Mar←Instr<28:31> next
        (decode (Instr<16:19> NEQ 0)@(Instr<12:15> NEQ 0)=>
        \00     RX0000:=      (NOP);                            ! No Base, No Index
        \01     RX00X2:=      (NOP);                            ! No Base, Indexing
        \10     RXB100:=      (NOP);                               ! Base, No Index
        \11     RXB1X2:=      (NOP)                                ! Base, Indexing
                ) next
        (if Instr<16:19> => Mar←Mar+R[Instr<16:19>]) next
        (if Instr<12:15> => Mar←Mar+R[Instr<12:15>]) next
        (decode Instr<2:7> =>
        . . . . . .                                          ! Select RX Instructions
                )
        end;
```

Figure 10 - Use of Artificial Labels

References

[AmdG64] Amdahl, G. M., Blaauw, G. A., and Brooks, F. P., "Architecture of the IBM System/360", IBM Journal of Research and Development, Vol. 8, No. 2, April 1964, pp. 87-101.

[AndV74] Anderson, V. L. and McLean, R. A., Design of Experiments, a Realistic Approach, Marcel Dekker, Inc., New York, 1974.

[BarM74] Barbacci, M.R. and Siewiorek D.P.: Some Aspects of the Symbolic Manipulation of Computer Descriptions. Department of Computer Science, Carnegie-Mellon University, July 1974.

[BarM75] Barbacci, M.R.: "A Comparison of Register Transfer Languages for Describing Computers and Digital Systems". IEEE Transactions on Computers, Volume C-24, Number 2, February 1975, pp. 137-149.

[BarM76a] Barbacci, M.R.: "The Symbolic Manipulation of Computer Descriptions: ISPL Compiler and Simulator". Technical Report, Department of Computer Science, Carnegie-Mellon University, 1976.

[BarM76b] Barbacci, M.R, D.P. Siewiorek, R. Gordon, R. Howbrigg, and S. Zuckerman: "Architecture Research Facility: ISP Descriptions, Simulation, Data Collection." Volume IV of Computer Family Architecture Selection Committee Final Report. Naval Research Laboratory, Washington D.C., December 1976.

[BelC71] Bell, C. G. and A. Newell, Computer Structures: Readings and Examples, McGraw-Hill, New York, 1971.

[BerN75] Bernwell, N. (editor), Benchmarking: Computer Evaluation and Measurement, John Wiley & Sons, New York, 1975.

[BoxG64] Box, G. E. P. and Cox, B. R., "An Analysis of Transformations", The Journal of the Royal Statistical Society, Series B, Vol. 26 (1964), 211-252.

[GMLC75] Computer Review (formerly Computer Characteristics Review, GML Corporation, Lexington, MA, 02173, 1975.

[ConW59] Connor, W. S. and Zelen, M., "Fractional Factorial Experiment Designs for Factors at Three Levels", National Bureau of Standards, Applied Mathematics Series Vol. 54, 1959.

[CorJ77] Cornyn, J.J, Smith, W.R., Svirsky, W.R., and Coleman, A.H.: "Two Life-Cycle Cost Models for Comparing Computer Architectures". Submitted to National Computer Conference, NCC-77.

[DavO71] Davies, O. L. (editor), Design and Analysis of Industrial Experiments, 2nd ed., Oliver and Boyd, Edinburgh, 1971.

[FulS76a] Fuller, S. H., Stone, H. S., and Burr, W. E., "Selection of Candidate Computer Architectures and Initial Screening." Volume II of Computer Family Architecture Selection Committee Final Report, Naval Research Laboratory, Washington, D.C. 20375. 1 December, 1976.

[FulS76b] Fuller, S.F., W.E. Burr, P. Shaman, and D. Lamb: "Evaluation of Computer Architectures via Test Programs". Volume III of Computer Family Architecture Selection Committee Final Report. Naval Research Laboratory, Washington D.C., 1 December 1976.

[FulS77a] Fuller, S. H., Burr, W. E., Shaman, P., and Lamb, D. A., "Evaluation of Computer Architectures via Test Programs." This Volume.

[FulS77b] Fuller, S.F., H.S. Stone, and W.E. Burr: "Initial Selection and Screening of the CFA Candidate Computer Architecture." This Volume.

[LucH71] Lucas, H. C., "Performance Evaluation and Monitoring", ACM Computing Surveys, 3, 3 (1971), pp 79-91.

[PopG74] Popek, G. J., and Goldberg, R. P., "Formal Requirements for Virtualizable Third Generation Architectures," Communications of the ACM, Vol. 17, No. 7, July 1974, 412-421.

[RaoC73] Rao, C. R., Linear Statistical Inference and its Applications, 2nd ed., John Wiley & Sons, New York, 1973.

[SmiW76] Smith, W.R., J.J. Cornyn, A.H. Coleman, W. Svirsky, R. Estell, P. Sabin: "Life Cycle Cost Models for Comparing Computer Family Architectures". Submitted to National Computer Conference, NCC-77.

[StoH75] Stone, H. S. (editor), Introduction to Computer Architecture, Science Research Associates, Chicago, 1975.

[StoH76] Stone, H. S., "An Audit of the Selection Criteria for Computer Family Architecture," CFA memorandum, January, 1976. Distributed at the 18-20 February CFA meeting.

[WagJ76] Wagner, J., B. Lieblain, J. Rodriguez, H.S. Stone: "Evaluation of the Candidate Architectures for the Military Camputer Family". Submitted to National Computer Conference, NCC-77.

[WicB73] Wichmann, B. A., Algol 60 Compilation and Assesment, Anderson Press, New York, 1973.

[WulW75] Wulf, W. et. al.: The Design of an Optimizing Compiler. American Elsevier, Programming Language Series, New York, 1975.