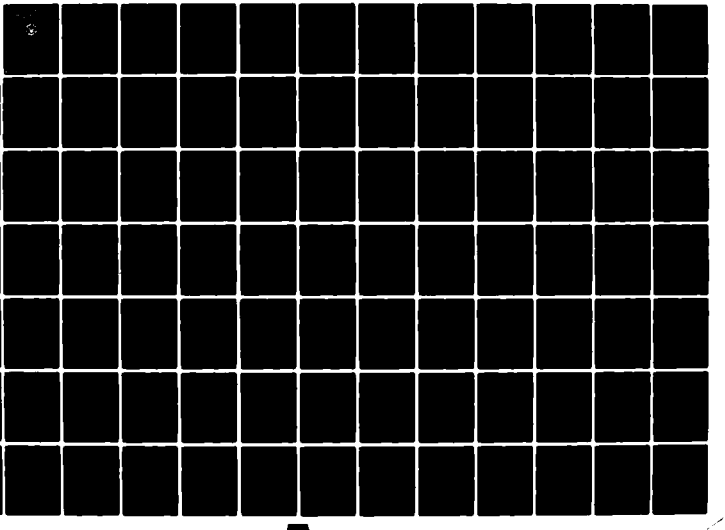
AD-A096 302 NAVAL POSTGRADUATE SCHOOL MONTEREY CA F/G 9/2
A COMPUTER EVALUATION TECHNIQUE FOR EARLY SELECTION OF HARDWARE--ETC(U)
DEC 80 B D HODGINS

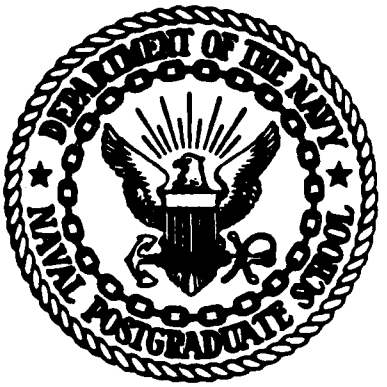
UNCLASSIFIED NL

1 OF 2
AD A096302



NAVAL POSTGRADUATE SCHOOL

Monterey, California



THESIS

A COMPUTER EVALUATION TECHNIQUE FOR EARLY SELECTION OF HARDWARE

by

Bart Dallas Hodgins

December 1980

Thesis Advisor: L. A. Cox, Jr.

Approved for public release; distribution unlimited

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A096 302	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) (6) A Computer Evaluation Technique For Early Selection of Hardware.	5. TYPE OF REPORT & PERIOD COVERED (9) Master's Thesis; December 1980
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) (10) Bart Dallas Hodgins	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940 (12) 124 (11)	12. REPORT DATE December 1980
	13. NUMBER OF PAGES 123
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Naval Postgraduate School Monterey, California 93940	15. SECURITY CLASS. (of this report) Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Hardware performance evaluation
Hardware selection
Instruction mix
Instruction mix sensitivity technique (IMSET)

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

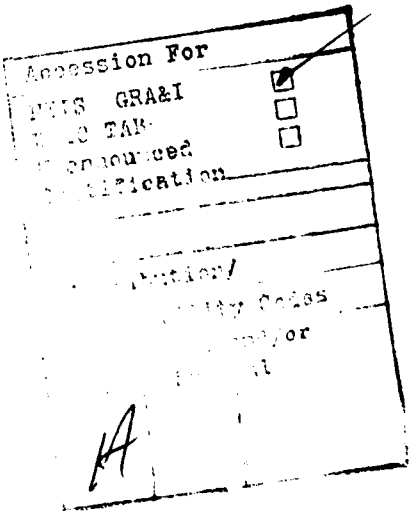
There is a need for a decision making/early selection tool for use in the government computer selection process. Such early selection tools are critical to the decision maker due to the environment in which the government procurer is forced to operate. The instruction mix sensitivity technique as demonstrated here has the potential to aid the government decision maker in evaluating the performance of a computer prior to the actual

DD FORM 1 JAN 73 1473 (Page 1) EDITION OF 1 NOV 65 IS OBSOLETE S/N 0102-014-6601

UNCLASSIFIED 251450

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

existence or availability of that hardware without resorting to costly and time consuming techniques such as simulation or modeling.



Approved for public release; distribution unlimited

A Computer Evaluation Technique For Early Selection of Hardware

by

Bart D. Hodgins
Lieutenant, United States Navy
B.S., University of Mississippi, 1973

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1980

Author: Bart Dallas Hodgins

Approved by: Tyler G. Cox Jr.
Thesis Advisor

Uno R. Kodres
Second Reader

Chairman, Department of Computer Science

W. M. Woods
Dean of Information and Policy Sciences

ABSTRACT

There is a need for a decision making/early selection tool for use in the government computer selection process. Such early selection tools are critical to the decision maker due to the environment in which the government procurer is forced to operate. The instruction mix sensitivity technique as demonstrated here has the potential to aid the government decision maker in evaluating the performance of a computer prior to the actual existence or availability of that hardware without resorting to costly and time consuming techniques such as simulation or modeling.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENT

I would like to thank Professor Lyle Cox. Without his patience and guidance this thesis would never have come to be. He was never reluctant to share with me his ideas and suggestions with which to make this major effort in my academic life a success. Through work on this thesis he became more than an advisor to me, he became a friend. A special thanks goes to my newlywedded wife, Suzanne, whose encouragement and typing skills helped to erase many of the darker hours of the past year.

I. INTRODUCTION

There is a need for a decision making/early selection tool for use in the government computer selection process. Such early selection tools are critical to the decision maker due to the environment in which the government procurer is forced to operate. The instruction mix sensitivity technique as demonstrated here has the potential to aid the government decision maker in evaluating the performance of a computer prior to the actual existence or availability of that hardware without resorting to costly and time consuming techniques such as simulation or modeling.

A. OPERATING ENVIRONMENT

Operating in our present U.S. Government environment, E.D.P. procurements evolve through a cycle that lasts five to seven years. The selection of computer hardware for use by the government is forced to occur early in the procurement cycle. This long time period from selection to operational installation often necessitates procurement decisions be made before prototype hardware is available. Hardware selections must be made quickly and accurately. Errors cost time and money. Any delay caused by selection will have a ripple effect building through the entire process causing larger delays before the system is realized at the operational level. The poor selection of the hardware to be used as the basis for a system

can result in cost overruns in other areas to compensate for the lack of acceptable hardware performance. These cost increases can be tremendous if the inadequate performance of the hardware must be compensated for in software.

At present there is no general method for computer hardware evaluation and selection suitable for use early in the procurement cycle. Given the requirement for early selection of hardware, poor procurements are often made because the decision maker is forced to make a selection without benefit of having candidate hardware (and/or software) available. Similarly, all too often the selections of equipments are based on imprecise and quantitatively vague ideas of the actual operational utilization the system will face in the future. It is not surprising that without an adequate method to evaluate this scanty information, mistakes will be made.

B. EARLY SELECTION PROBLEMS

There are several methods currently being utilized for the evaluation of a computer's performance. They include: (1) benchmark programs which are existing programs coded in a specific language, then executed and timed on a target machine [1], (2) kernel functions which are typical functions partially or completely coded and timed [1], (3) simulations which are a combination of a model of the system, model of the workload, and a measurement of the resulting data [2], and (4) analytic models which are mathematical representations of the target machine [1]. These methods are all in use by industry to

evaluate proposed computer systems for procurement. These methods are effective for civilian procurements because their operating environment is much different from that of the government. The industry procurement cycle may take less than one year. They are not required to make their selection early. By waiting until both hardware and software are available, industry is able to utilize the classic evaluation techniques in making a specific computer system selection.

The government buyer, forced to select early, is faced with unique problems that the various evaluation techniques can not solve. Evaluation by the benchmark program method is impossible because the various hardwares are not always available. Even if a prototype hardware of a future system were available for evaluation, the benchmark programs and the kernel function methods would prove inadequate to the government decision maker because the software required to validate the technique usually does not exist at that point. Validation insures that the benchmark programs and kernel functions accurately reflect the intended application. Without the software in existence, the validation of the benchmark and kernel function programs is impossible.

The government manager, being forced with a quick selection, has neither the time, money, or sufficient detailed design information necessary to model/simulate the proposed computer systems. It is because of this problem that the instruction mix sensitivity technique (IMSET) has been developed.

C. BASIS OF TECHNIQUE

The instruction mix sensitivity technique is based upon the older instruction mix method for predicting computer hardware performance. In the instruction mix method a number was computed which represented the average thruput of a particular hardware. This number was based upon the relative usage of a given instruction in a particular application, and its execution time on the evaluated hardware. Where the older method was based on a single mix representing a specific application, the sensitivity technique uses differentials between a collection of mixes representing various applications. The advantage of this technique is that neither the hardware of software need be completed--only the organization and technology need be determined. The eventual utilization of the system need not be precisely defined. This technique provides immediate evaluation results with a minimum expenditure of time and money.

Using the IMSET requires only that the vendor furnish the performance specifications regarding instruction execution times. These performance specifications are often available years in advance of a prototype model. With these times, and the analysis technique presented here, the evaluator can evaluate the performance of any hardware against the anticipated application. The particular machines to be considered in the selection need not be prototyped.

The use of the IMSET as a tool for evaluation provides the decision maker with a profile representing the candidate computer's average execution time for the various applications presented in the set of instruction mixes. From the data obtained for a computer the decision maker can select the hardware with the best profile for the mix(s) matching general areas of intended application. For example, if the evaluator is looking for a machine to perform accounting functions then the selection would be based upon how sensitive each candidate is to the mixes which represent accounting functions. The less sensitive the machine in terms of execution time the more appropriate it would be for selection, since this indicates that it can execute effectively a broad spectrum of related functions.

Section Two presents a brief history of Computer Performance Evaluation and the instruction mix technique in particular. Section Three deals with the development and use of the instruction mix sensitivity technique as a tool for selection and evaluation. Section Four presents a demonstration using the IMSET in the evaluation of a broad range of known and existing computer hardware including maxis, minis, and micro-computers. Section Five presents conclusions and recommendations for future development and use of IMSET.

II. HISTORY OF COMPUTER PERFORMANCE EVALUATION

The instruction mix as a technique for evaluating the performance of a computer's hardware came into being in the late 1950's and early 1960's. It evolved as a result of the limitations of an earlier technique for measuring a computer's performance called the instruction execution timing method. This technique, sometimes called the "cycle-add" technique, was used to compare memory cycle times and arithmetic instruction execution times, normally the ADD or MULT instruction of given CPU's. This method was at the time considered adequate because operating systems and compilers were as of yet unheard of, and what assemblers were available were very crude. All programs were written directly for the hardware. Under these circumstances, the cycle-add times reflected machine capabilities fairly well.

Machine architectures began to change as technological advancements lowered the costs of memory units and periphal devices. The development of software support packages consisting of operating systems, compilers, and assemblers hastened to make computer systems more complex. These advancements led to special features being introduced into computer designs. Features such as parallelism, pipelining, and compound addressing, added power while decreasing the execution times of individual instructions. These changes made evaluation

by the cycle-add method extremely unreliable. The method did not account for the organizations of the new machines being produced (i.e. input/output, multi-address instructions, etc.). It similarly failed to assess the impact of the new monitors, assemblers, and compilers which were non-numeric programs running on the machines being evaluated. The impact upon system performance due to these non-numeric programs was impossible to assess with the cycle-add method. It was because of these shortcomings that the instruction mix technique as a performance evaluation tool evolved.

A. INSTRUCTION MIX TECHNIQUE

The instruction execution timing method incorporated only the arithmetic class of instructions. The instruction mix technique incorporated along with the arithmetic class, the logical class (i.e. COMPARE, AND, OR, etc.), the control class (i.e. BRANCH, SHIFT, MOVE, etc.), and in some instances I/O and other miscellaneous instructions. Associated with each instruction in the mix was a percentage of use of that instruction, called a weighting factor unique to that particular mix. This weighting factor represented the approximate probability of occurance of that instruction in the programs to be used on the machine. For instance, in a scientific instruction mix one would find that the percentage of floating point multiplications would be higher than the percentage for that same instruction in the data processing instruction mix. Table I

shows two typical instruction mixes with their associated weight functions for each instruction type included in the mix.

The probabilities in an instruction mix are normally determined by either statically or dynamically tracing the programs representing a specific application. This determins the relative frequency of use of the different types of instructions in an application. The dynamic method is preferred over the static method because the static trace does not take into account multiple executions of loops. The dynamic trace, counting instructions as they are executed, takes multiple executions into account, but is more difficult and expensive.

The instruction mix technique is easy to apply. By multiplying the execution time of each instruction by the weighting factor and summing, one obtains the average time required to execute an instruction for that particular mix on that particular computer. This average time can be expressed as a thruput rate in kilo-instructions-per-second (KIPS). These totals can then be compared with similar rates obtained from other machines, to give an idea of relative CPU thruput. For a sample thruput comparison refer to Table II.

This method gained immediate popularity because of its ease of use and because it could be based upon easily acquired data. As a result instruction mixes for many applications

TABLE I

INSTRUCTION MIXES WITH WEIGHT FUNCTIONS

		Gibson	Navigation
I.	Arithmetic		
	A. Fixed Point (SP)		
	1. Add/Sub (RR)	0.061	0.23
	2. Multiply (RR)	0.060	0.25
	3. Divide (RR)	0.020	0.00
	B. Fixed Point (DP)		
	4. Add/Sub (RR)	0.000	0.00
	5. Multiply (RR)	0.000	0.00
	C. Floating Point (SP)		
	6. Add/Sub (RR)	0.000	0.00
	7. Multiply (RR)	0.000	0.00
	8. Divide (RR)	0.000	0.00
II.	Logical		
	9. Compare (RX)	0.038	0.02
	10. Shift (8 bits)	0.044	0.00
	11. And/Or	0.016	0.00
III.	Control		
	12. Load/Store	0.312	0.30
	13. Branch Conditional	0.166	0.02
	14. Branch Unconditional	0.000	0.00
	15. Inc & Store Index	0.180	0.04
	16. Move (RR)	0.053	0.00
	17. Index	0.000	0.00
IV.	I/O & Miscellaneous		
	18. I/O & Miscellaneous	0.050	0.14

Note: Where zeros are indicated, weights were not assigned by the mix for the indicated functional instruction.

TABLE II

CPU THRUPUT CALCULATION: INSTRUCTION MIX METHOD

	Gibson	AN/AYK-14(V) (μsec)	(Weight)x(Time)
I. Arithmetic			
A. Fixed Point (SP)			
1. Add/Sub (RR)	0.061	0.90	0.0549
2. Multiply (RR)	0.060	4.20	0.0252
3. Divide (RR)	0.020	8.30	0.0166
B. Fixed Point (DP)			
4. Add/Sub (RR)	0.000	1.10	0.00
5. Multiply (RR)	0.000	6.90	0.00
C. Floating Point (SP)			
6. Add/Sub (RR)	0.000	4.00	0.00
7. Multiply (RR)	0.000	5.00	0.00
8. Divide (RR)	0.000	56.10	0.00
II. Logical			
9. Compare (RX)	0.038	2.00	0.0760
10. Shift (8 bits)	0.044	4.10	0.1804
11. And/Or	0.016	0.90	0.0144
III. Control			
12. Load/Store	0.312	1.95	0.6084
13. Branch Conditional	0.166	1.90	0.3154
14. Branch Unconditional	0.000	1.90	0.00
15. Inc. & Store Index	0.180	1.40	0.2520
16. Move (RR)	0.053	0.90	0.0477
17. Index	0.000	2.10	0.00
IV. I/O & Miscellaneous			
18. I/O & Miscellaneous	0.050	5.04	0.2520
		Total:	1.8430 sec

KIPS = 1000/1.8430μsec
KIPS = 542.59

were developed. The most popular of all mixes was the Gibson Mix [3] developed by Jack C. Gibson in 1959 on data obtained on the IBM 7090 computer. The Gibson Mix was considered a general-technical mix. There were other similar mixes [4, 5, 6, 7, 8, 9, 10] for data processing, navigation, scientific, and a myriad of other applications. The instruction mix technique represented a tool which was quick and simple to use in the context of intended applications when comparing hardwares for selection and evaluation, or for designing new processors.

As computers continued to advance with increasing technology in both hardware and software, and as systems moved into a multiprogramming environment, it soon became apparent that the instruction mix technique as a method for evaluating performance was no longer adequate. Among its shortcomings was its failure to account for differences in addressing modes, word sizes, and operand lengths. The effects of I/O was still virtually ignored. Compilers and special features of individual CPU's made it difficult to validate the mix weights assigned to each instruction. The effect of system software upon the mix weights was difficult to assess.

Perhaps the biggest disadvantage was the problem of how to validate an instruction mix to insure that a particular mix accurately reflected the intended application. A scientific application coded by one person may have many instances of the DIVIDE instruction, whereas another programmer may use

very few, if any, DIVIDE instructions, but many MULTIPLY instructions. In this case does the scientific instruction mix still accurately reflect the application? Further, how can the instruction probabilities be determined if the programs representing the eventual workload have not yet been written?

B. BENCHMARK PROGRAM

In the search for a better method to replace the instruction mix the benchmark program technique was developed. The benchmark method is simply a program, or a collection of selected programs, coded in a specific language, to represent the typical workload of the system to be evaluated. The goal is to exercise, by a series of sequence calls, all systems software functions such as job schedules, file management, I/O support, and language processors. In this way the evaluated computer's multiprogramming/multiprocessing operating system is tested. The benchmark programs are executed a number of times on the computers being evaluated, and then the average execution times are compared.

Benchmark programs helped to eliminate some of the drawbacks that the instruction mix technique exhibited. However, the benchmark program method has its own drawbacks when used in the selection and evaluation environment. One problem is essentially identical to the validation problem associated with the instruction mix technique: how does one know the benchmark programs accurately reflect the future workload of the system? Second, since benchmark programs are real jobs

they often require a large conversion effort to interchange benchmark programs between systems. This process is time consuming and expensive. The biggest problem is that the benchmark program technique requires that the hardware and operating software all be available for testing, because compilers and their effects have an impact on the hardware execution times. The benchmark as a tool for selection and evaluation was well received when it was introduced. It is still used as a selection tool today in many commercial contexts. It is extremely useful in that it can be used as a before and after test to monitor performance following a change to an existing system.

C. KERNEL FUNCTION

An evaluation method similar to the benchmark program is the kernel function method. In this method a program consisting of a central or key function is either partially or completely coded and timed based upon the manufacturer's specifications for execution times. Examples of kernel functions are polynomial evaluations, matrix operations, report formating, table lookups, and comparison and sorting operations. The kernel differs from the benchmark programs in that the benchmarks are actually coded and executed, while kernels are not executed. The kernels can be designed to utilize all features thought to be necessary. This technique does consider differences in addressing logic and special index registers which the instruction mix method ignored. However, many

of the disadvantages common to the instruction mix method are likewise common to the kernel function method. The kernel function method, as the instruction mix before it, fails to completely consider I/O operations. Kernels can be biased: designed to make a given CPU look either good or bad. Validation of kernel functions remains a problem.

D. SIMULATION

Perhaps the most flexible and complete tool available today for evaluating computer performance is simulation. This method required the creation of models of the elements of a given system, including the system workload, and the process interactions occuring within the system. The simulator behaves as specified by the functional, and workload models in an identical manner as the simulated system would respond. The simulator collects performance data necessary for the evaluation.

There are a number of problems with simulation models. When using simulation methods the level of detail in the model is critical. Too little detail and the simulations results can be unreliable. Too much detail and the simulation becomes too costly for development and use. Additionally, with detailed simulations the run time is long and variations occur that make certain general aspects of the system's behavior hard to identify. Development of workload models are diffcult to validate. Complete hardware models are lengthy and error prone. Additionally, simulations are difficult to generalize and simulator systems are typically not portable.

Excellent results have been obtained by using simulation for selection evaluation. It allows the system to be studied under known conditions and controls. However, the simulation itself is its biggest disadvantage. It is extremely expensive to develop. The time, effort, and cost required to develop an accurate simulation model is usually well beyond the resources of a normal procurement effort. However, in situations such as development and design efforts, given sufficient budget and time, evaluation by simulation is an efficient alternative to building prototypes.

E. ANALYTIC MODELS

Performance evaluation by use of an analytic model involves mathematically representing the system to be evaluated [1, 2]. Such models normally are used to evaluate performance of a particular system management resource such as CPU scheduling, or file organization [2].

Analytical models are useful as additional points of reference in hardware analysis when used in conjunction with other evaluation methods.

These models require revision when moved from one hardware to another which increases the amount of time and cost involved over and above the original effort that went into initial development.

F. CHOICE OF EVALUATION METHOD

The methods for performance evaluation presented here have at one time or another received wide popularity. Each

had its unique attractions and limitations. Which method should one use is the question facing the evaluator. This decision must be based upon the constraints placed upon the decision maker by the procurement requirements and limitations. With a large budget and no time constraints, simulation is the most reliable method for selection. If one has a minimal budget, a reasonable amount of time to make the selection, and the candidate machines are available, then the benchmark method may be appropriate. If one is tightly constrained by time, or if the candidate machine prototypes have not yet been assembled, then the instruction mix technique would be the logical alternative if its shortcomings could be resolved.

The following section will discuss how the instruction mix sensitivity technique resolves these problems and can be used in a wide variety of critical selection situations.

III. TECHNIQUE FOR EARLY SELECTION

The technique for computer hardware evaluation and selection that is presented in this thesis is based upon the instruction mix method. It is contended that the various disadvantages mentioned in previous sections can be overcome to provide an efficient tool that the government decision maker can utilize. In this section the disadvantages and proposed solutions will be discussed.

For clarity of understanding, it must be pointed out that the instruction mix is a tool to be used principly for the comparative evaluation of the central processor hardware. The way the central processor is configured with other system components such as storage devices and other I/O and peripheral devices must be considered separately. The software associated with the system which includes the operating system, language processors, and applications programs also have an impact upon overall performance; however selection of this type of software is outside the scope of this work. By beginning the selection of a computer system with an appropriate central processor, the remaining decisions regarding peripherals and software are made much easier.

A. DISADVANTAGES OF INSTRUCTION MIX TECHNIQUE

The basic disadvantages of the instruction mix technique are: (1) difficulties in accounting for the number of operands

per instruction, (2) differences in addressing modes used within a given machine, (3) the number of instructions needed to code the same task on different machines varies, (4) instructions vary between machines, (5) word lengths are unequal between machines, (6) machine overlap capabilities are ignored, (7) I/O instructions are omitted in many instruction mixes, and (8) validation of particular mixes is not assured. Taken as a whole these disadvantages are significant and in many contexts preclude the use of the instruction mix technique. The variation of the instruction mix technique presented in this thesis will diminish the significance of some of the disadvantages, and eliminate others altogether.

B. INSTRUCTION MIX SENSITIVITY TECHNIQUE (IMSET)

The variation of the instruction mix technique presented here is called the instruction mix sensitivity technique (IMSET). The IMSET uses a set of ten instruction mixes chosen from an original twenty-two candiate mixes. These mixes represent all aspects of computer applications, spanning from real-time computations thru scientific to business processing. The method of selection is explained in the following section. Utilization of the IMSET provides the evaluator with a profile representing a hardware's execution times across all mixes in the set (and hence a broad spectrum of applications). The profile of execution times provides the decision maker with an evaluation of how sensitive each computer is to the various mixes and hence how the system will perform over a wide range

of applications the system is likely to face in the future. This is in contrast to the instruction mix technique which only provided the evaluator with a thruput evaluation on one mix--one application.

The significance of this difference is critical. The final ten mixes which are included in the IMSET were determined through extensive evaluation as to the amount of significant information they were actually presenting. The mixes that were eliminated were found to present no new information. Those mixes that remain provide the decision maker with the smallest number of mixes which preserved the maximum amount of vital information over the complete range of applications. Their use shows how sensitive a CPU is to various applications. This is especially important when the ultimate use of the computer is not precisely known at evaluation time. This is in contrast to the instruction mix technique which provides one evaluation for one specific application.

The IMSET developed in this thesis uses eighteen functional instructions which constitute the basis for evaluation. These include seventeen specific instructions and one I/O miscellaneous category. There is no instruction mix that provides a weight function for all eighteen instructions listed, but taken as a group all instructions listed are covered at least once by a mix. The eighteen functional instructions and ten mixes which constitute the IMSET are shown in Table III.

TABLE III

IMSET FUNCTIONAL INSTRUCTIONS AND MIXES

	I. ARITHMETIC								II. LOGICAL			III. CONTROL						IV. I/O MISC.
	FIXED POINT (SP)			FIXED POINT (DP)		FLOATING POINT (SP)							BRANCH					
IMSET MIXES	ADD/SUB	MULTIPLY	DIVIDE	ADD/SUB	MULTIPLY	ADD/SUB	MULTIPLY	DIVIDE	COMPARE	SHIFT	AND/OR	LOAD/STORE	CONDITIONAL	UNCONDITIONAL	INC & STORE INDEX	MOVE	INDEX	I/O & MISC.
PROCESS CONTROL	.064	.013	.001	0.0*	0.0*	0.0*	0.0*	0.0*	0.0	.022	0.0	.400	.430	0.0*	0.0	0.0	0.0*	.020
MESSAGE PROCESSING	.050	.005	.005	0.0*	0.0*	0.0*	0.0*	0.0*	.010	.030	.150	.470	.140	0.0*	.030	.058	0.0*	.052
REAL TIME	.126	.108	.020	.014	.012	0.0*	0.0*	0.0*	.020	.070	.010	.500	.100	0.0*	0.0*	0.0*	0.0*	.020
COMMUNICATION CNTL	.080	.002	.002	0.0*	0.0*	.005	.001	.001	.075	.075	.070	.475	0.0*	.050	.030	0.0*	.035	.090
DATA COMPRESSION	.190	.060	.060	0.0*	0.0*	0.0*	0.0*	0.0*	.120	0.0	0.0	.270	.090	0.0*	.060	.060	0.0*	.120
NAVIGATION	.230	.250	0.0	0.0*	0.0*	0.0*	0.0*	0.0*	.020	0.0	0.0	.300	.020	0.0*	.040	0.0	0.0*	.140
TLM THRUPUT	.090	.040	0.0	0.0*	0.0*	0.0*	0.0*	0.0*	.290	0.0	0.0	.220	.150	0.0*	.040	0.0	0.0*	.190
TECHNICAL GENERAL	.025	.025	.025	0.0*	0.0*	.011	.011	.011	.108	.045	0.0*	.361	.275	0.0*	0.0*	0.0*	0.0*	.103
SCIENTIFIC	0.0*	0.0*	0.0*	0.0*	0.0*	.095	.056	.020	0.0*	0.0*	0.0*	.285	.132	0.0*	0.0*	0.0*	.225	.187
COMPOSITE GENERAL	.022	.022	.022	0.0*	0.0*	.003	.003	.003	.120	.029	0.0*	.361	.359	0.0*	0.0*	0.0*	0.0*	.057

* Weight not assigned by mix for this functional instruction.

Use of the IMSET is now simply a matter of determining the execution times of each instruction indicated for each candidate central processor hardware to be evaluated. The time to execute each mix is then determined by use of the following formula:

$$TE = \sum_{i=1}^{n} I_i \, M_i \qquad (1)$$

where

$$\sum_{i=1}^{n} I_i = 1 \qquad (2)$$

and,

TE: time to execute a particular mix

I_i: instruction weight

M_i: machine's time to execute instruction indicated

n: number of functional instructions being considered for evaluation (in this thesis 18)

With the computed TE's, the decision maker is then able to compare processors either as a raw total, or as a ratio of two processors's TE's. A computational example is given in Section Four.

C. RESOLVING THE PROBLEMS OF INSTRUCTION MIX TECHNIQUE

When applying an evaluation technique it is necessary to make certain assumptions. One basic assumption of the IMSET is that principally the central processor and arithmetic hardware is being evaluated for selection. For this reason, all

of the specific instructions identified in a particular mix are taken as register-to-register operations, except for the LOAD/STORE which will require a register to-memory operation. For instance, a mix's fixed point ADD instruction is taken to mean ADD R1, R2 in a two address machine, rather than ADD X, Y where X and Y are memory addresses. This is the time taken from a particular machine's array of ADD times in its instruction set for use in IMSET. All other ADD times are then lumped together as an average time, along with the average of the times of all instructions not used in the mix calculations, to form the category of "miscellaneous instructions". By assuming the same operations, in the arithmetic case register-to-register, across all machines being evaluated, (where possible) the number of operands to be accounted for is not a problem.

The problem of different addressing modes within a given machine is solved by taking the average time for that instruction to execute all modes. (Appendix C gives examples of this using the PDP 11/70.) It is realized that different programmers and language processors will generate code in different ways; however, at this level of detail the average is an acceptable approximation.

When a machine does not have an instruction in its set which will perform a task specified in one of the selected mixes, then more than one instruction must be used to accomplish this task. Examples of this occur with the microcomputers

in Appendix C. It is true that the number of instructions to accomplish this task will vary from machine to machine but that is precisely what the evaluator is looking for in an evaluation. The evaluator wants to know that a tremendous time penalty must be paid if an INTEL 8080 processor is selected with the idea of doing scientific calculations, since this processor has no floating point instructions and must simulate these functions with subroutines.

Machines with unequal word lengths are no longer as significant problem for the evaluator as it was 15 years ago. When evaluation time comes the minimum acceptable word length must be determined, and comparisons made on this basis. Functions in the instruction mixes can be defined in terms of the necessary precision. For example, the MULT instruction can be defined as the time to complete a 32-bit multiply, or a 16-bit multiply, whichever is appropriate.

In the standard application of the instruction mix technique many special features of a central processor's hardware were ignored. The most important feature being ignored was the ability to overlap instructions. The overlap feature allows a central processor to begin execution of a second instruction before the current instruction has finished its execution. This allows effective execution times to be cut significantly. The IMSET presented here takes into account the overlap capabilities of the machines being evaluated by applying a "Knuth Factor". This idea was provided by [11].

The Knuth Factor compensates for the machines which have overlap or parallel processing abilities by scaling down their execution times by an amount comparable with the use of this feature typical of most compilers. It is based upon the idea that the "smarter" the compiler the greater is its ability to provide a compiled program capable of taking advantage of CPU parallelism. For example, the CDC 6600 utilizes ten functional units which provide instruction execution. If one of the functional units, for instance the ADD unit, is in execution, and the next instruction is an ADD instruction, then the CPU must wait until the ADD unit is free. An optimal compilation of a CDC 6600 program would try to re-arrange two or more instructions requiring the same functional unit, so that they would not occur together. How the Knuth Factor was determined and how to apply it is presented in Appendix D with examples of its use in Appendix C. The machines presented in the demonstration of the IMSET which have overlap capabilities have the Knuth Factor applied to them, and the resulting execution times for any particular mix shows a significant time savings.

The I/O instructions omitted from many mixes caused problems with early evaluations. I/O instructions are a mixture of peripheral capability and a central processor capability. The mixes presented here include the I/O instructions in the miscellaneous category rather than as a specific instruction. In this way the central processor's ability to handle I/O is

treated as an average over all of the I/O instructions without having to specify an exact instruction or particular device.

Validation of the mixes vs. applications when using the IMSET for evaluation is not the problem it was for the instruction mix method. When evaluating by the IMSET method, particular sensitivities within a broad area of intended use are being measured, whereas with the instruction mix method, execution time of a specific application was being estimated. Thus, validation is not a problem when utilizing the IMSET.

The advantages to using the IMSET over other currently used techniques are tremendous. As mentioned in previous sections, government evaluators work in a completely different environment than their civilian counterparts. Government selectors are not able to utilize many of the more sophisticated, and proven methods. With the IMSET presented here the decision maker needs only the manufacturer projected instruction set execution times. With these times the decision maker can obtain the evaluation data within a matter of hours and at minimal cost. This technique provides a savings in time, savings in money, greater confidence in the selection, and perhaps its most attractive advantage, is its ease of use.

D. DEVELOPMENT OF IMSET

The IMSET evolved through a two-stage process. The initial stage of the process consisted of six steps:

(1) selecting numerous mixes covering a variety of applications, (2) determing which functional instructions to include, (3) choosing the machines with which to evaluate the mixes, (4) determing each machine's instruction execution times, (5)conducting demonstrations of machines vs. mixes, and (6) analyzing the data resulting from the demonstration to determine which mixes presented redundant information and thereby should be eliminated from the final evaluation stage. The final stage of the IMSET process consisted of four steps: (1) choosing new machines which to evaluate and test the IMSET, (2) determing instruction execution times for each machine, (3) obtaining profiles for each machine, and (4) presenting and analyzing profile results.

1. Initial Stage

a. Selection of Mixes

Twenty-two mixes were gathered from a variety of sources. All are presented in Table VIII in Appendix A. Of the original twenty-two, two were quickly eliminated from further investigation because of their lack of completness (Knight scientific mix, and the Knight commercial mix). The twenty remaining mixes, shown in Table IV, covered a broad range of applications with many applications being represented by more than one mix. These twenty mixes served as the basis for further study.

b. Functional Instruction Determination

Analysis of the mixes determined which functional instructions would be used to evaluate hardware performance.

TABLE IV

INITIAL DEMONSTRATION MIXES

	I. ARITHMETIC								II. LOGICAL			III. CONTROL						IV. I/O MISC.
	FIXED POINT (SP)			FIXED POINT (DP)		FLOATING POINT (SP)							BRANCH					
	ADD/SUB	MULTIPLY	DIVIDE	ADD/SUB	MULTIPLY	ADD/SUB	MULTIPLY	DIVIDE	COMPARE	SHIFT	AND/OR	LOAD/STORE	CONDITIONAL	UNCONDITIONAL	INC & STORE INDEX	MOVE	INDEX	I/O & MISC.
PROCESS CONTROL	.064	.013	.011	0.0*	0.0*	0.0*	0.0*	0.0*	0.0	.002	0.0	.400	.480	0.0*	0.0	0.0	0.0*	.020
MESSAGE PROCESSING	.050	.005	.005	0.0*	0.0*	0.0*	0.0*	0.0*	.010	.030	.150	.470	.140	0.0*	.030	.058	0.0*	.052
REAL TIME	.126	.108	.020	.014	.012	0.0*	0.0*	0.0*	.020	.070	.010	.500	.100	0.0*	0.0*	0.0*	0.0*	.020
COMMUNICATION/CNTL.	.080	.002	.002	0.0*	0.0*	.005	.003	.001	.075	.075	.070	.475	0.0*	.050	.030	0.0*	.035	.090
DATA COMPRESSION	.190	.060	.060	0.0*	0.0*	0.0*	0.0*	0.0*	.120	0.0	0.0	.270	.060	0.0*	.060	.060	0.0*	.120
NAVIGATION	.230	.250	0.0	0.0*	0.0*	0.0*	0.0*	0.0*	.020	0.0	0.0	.300	.020	0.0*	.040	0.0	0.0*	.140
TIM THRUPUT	.090	.040	0.0	0.0*	0.0*	0.0*	0.0*	0.0*	.290	0.0	0.0	.220	.150	0.0*	.040	0.0	0.0*	.190
TECHNICAL GENERAL	.625	.025	.025	0.0*	0.0*	.011	.011	.011	.100	.045	0.0*	.361	.275	0.0*	0.0*	0.0*	0.0*	.103
SCIENTIFIC	0.0*	0.0*	0.0*	0.0*	0.0*	.695	.050	.020	0.0*	0.0*	0.0*	.280	.132	0.0*	0.0*	0.0*	.225	.187
COMPOSITE GENERAL	.022	.022	.022	0.0*	0.0*	.003	.003	.003	.120	.029	0.0*	.361	.358	0.0*	0.0*	0.0*	0.0*	.057
E D P	.126	.023	.009	0.0*	0.0*	0.0*	0.0*	0.0*	.106	.071	.035	.337	.200	0.0*	.031	.017	0.0*	.045
RADAR DATA PROC.	.066	.018	.003	0.0*	0.0*	0.0*	0.0*	0.0*	.100	.150	.112	.177	.238	0.0*	.035	.090	0.0*	.002
CONTROL/DISPLAY	.080	.006	.002	0.0*	0.0*	0.0*	0.0*	0.0*	.007	.050	.060	.470	.220	0.0*	.090	.010	0.0*	.005
COMMUNICATION/CNTL.	.085	.005	.002	0.0*	0.0*	0.0*	0.0*	0.0*	.065	.075	.070	.475	.125	0.0*	.065	.010	0.0*	.023
TRACK & COMMAND	.180	.050	.020	0.0*	0.0*	0.0*	0.0*	0.0*	0.0	.070	.030	.380	.200	0.0*	0.0	.070	0.0*	0.0
RADAR SEARCH TRACK	.053	.013	.004	0.0*	0.0*	0.0*	0.0*	0.0*	.041	.074	.049	.485	.241	0.0*	.014	.005	0.0*	.021
GIBSON	.061	.006	.002	0.0*	0.0*	0.0*	0.0*	0.0*	.038	.044	.016	.312	.160	0.0*	.180	.053	0.0*	.050
REAL TIME	.160	.050	.020	0.0*	0.0*	0.0*	0.0*	0.0*	.120	.050	.040	.330	.100	0.0*	.040	.050	0.0*	.040
GENERAL PURPOSE	.107	.036	.012	0.0*	0.0*	0.0*	0.0*	0.0*	.086	.051	.045	.337	.184	0.0*	.048	.041	0.0*	.053
COMMUNICATIONS	.150	0.0*	0.0*	0.0*	0.0*	0.0*	0.0*	0.0*	0.0*	.050	.050	.400	.150	.150	0.0*	0.0*	0.0*	.050

* Weight not assigned by mix for this functional instruction.

The first seventeen instructions of the IMSET were selected by examining all candidate mixes, and choosing those instructions which represented basic operations. The remaining instructions were combined into the I/O-Miscellaneous category which made up the eighteenth function instruction. Within the I/O Miscellaneous group are instructions such as, PROGRAMMED I/O TRANSFER, INTERRUPT RESPONSE, INITIALIZE BUFFERED I/O, and each mix's MISCELLANEOUS/OTHER category. For the specific instructions to be used in the IMSET for hardware evaluation refer to Table III.

c. Machine Selection

The computer hardwares to be evaluated in this stage of the demonstration were selected because of their differences in speeds and organizations (i.e. bus structure, functional units, floating point hardware, etc.). This was intended to give the technique a broad range of input so that the amount of information gathered from the mixes could be assessed. This information was then used for a correlation analysis to determine which mixes could be eliminated as previously mentioned. The computers choosen for this stage of the development are listed in Table V.

d. Instruction Execution Times

The determination of the machine instruction execution times for each CPU is presented in Appendix C. A number of these machines utilize special features which decrease their overall execution times. The PDP 11/70 utilizes

TABLE V

HARDWARES CHOSEN FOR INITIAL DEMONSTRATION

COMPUTER	TYPE
PDP 11/70	Mini
IBM 360/30	Maxi
IBM 360/75	Maxi
CDC 6600	Maxi
CRAY 1	Maxi
HONEYWELL LEVEL-6/43(1)	Mini
AN/UYK-20	Mini
AN/UYK-7	Maxi
AN/AYK-14(V)	Mini

(1) Also known as AN/UYK-37

a Floating Point Processor (FPP) for it's floating point instructions. Honeywell Level-6/43 uses a Scientific Instruction Processor (SIP) for the same purpose. The CDC 6600 and the CRAY 1 both have functional units which execute instructions sent to them by their respective CPU's. These features provided by the various hardwares allow for the execution of a number of instructions simultaneously. This parallel processing ability has been taken into consideration. Each applicable instruction of each machine processing these execution enchancements has been scaled by the "Knuth Factor" previously described.

e. Initial Stage Demonstration

The actual evaluation was computerized and run on a PDP 11/50 with graphics output. Each computer listed in

Table V was evaluated over the twenty mixes listed in Table IV.

f. Analysis and Determination of Final Mixes

The results obtained from the demonstration were run through the IBM 360/65 utilizing the statistical software package, SPSS. The mean, variance, and range of each mix was then computed. Each mix was then compared with each of the other mixes to detect correlations. By ranking the correlation data obtained for each pair of mixes from highest correlated to least correlated, and then taking a frequency count of mixes in highly correlated pairs, mixes which contained redundant information were identified and discarded. The mixes providing the greatest amount of information are listed in Table III. These ten mixes form the basis of the IMSET. Section Four provides typical profiles, Figures 1 through 24, for all hardwares presented in this thesis.

IV. DEMONSTRATION OF IMSET

A. FINAL STAGE

1. Machine Selection

In the final stage of the IMSET development process six micro-computers were selected. These were evaluated along with the original nine computers chosen during the initial stage. The introduction of the micros was done to accent the strength of the IMSET when used to evaluate machines closely related in characteristics. This demonstration would more accurately reflect an actual evaluation for selection situation which a government procurer would be facing. The micros selected are all 8-bit or 16-bit machines ranging from some earlier models to some much more recent ones. Those selected are presented in Table VI.

TABLE VI

PROCESSORS

ZILOG 8000

INTEL 8086

MOTOROLA 68000

INTEL 8080

DIGITAL EQUIP CORP LSI 11/23

TEXAS INST. 9900

2. Instruction Execution Times

Determination of the individual instruction times for each of the micro-computers is presented in Appendix C. Accounting for parallel processing capabilities by use of the Knuth Factor for an individual micro-computer was not necessary. As none of the micros have parallel processing capabilities.

A major factor to be considered and resolved when determining instruction execution times of micros is that of determining an appropriate algorithm to account for an instruction in the IMSET which is not part of the processor's instruction set. For instance, many of them do not include floating point instructions as part of their set. (An even worse case was the INTEL 8080 which does not have a fixed point multiply or divide instruction.) Resolving these difficulties involves some careful thought as to how a floating point operation or a fixed point multiply and divide is actually accomplished, and then providing a software routine to accomplish the task.

The absence of floating point instructions proved to be an easy task to resolve. A floating point ADD would be estimated by two fixed point ADD's and five shifts; a floating point SUB would be two fixed point SUB's and five shifts; a floating point MULT would be a fixed point ADD of the exponents, a fixed point MULT of the mantissas, and ten shifts for normalization; a floating point DIV is one fixed point SUB of the

exponents, one fixed point DIV of the mantissas, and ten shifts to normalize.

Determining fixed point multiply and divide routines for the INTEL 8080 was a much more involved task. The algorithm used to determine the multiplication execution time was based upon the example for fixed point multiplication in [12, pg. 138-139]. For fixed point division see [12, pg. 142-143]. Both of these algorithms were coded into 8080 assembly language, and the timing information was taken directly from ref. [13]. It may be contended that there are faster algorithms available for 8080 execution of these two instructions, but the versions used are representative.

3. Final Stage Demonstration

The final demonstration to obtain the profiles of all hardwares chosen versus the set of ten mix applications of the IMSET was conducted on the computerized evaluation system. The profiles are shown in Figure 1 through Figure 24. Figures 1-9 presents the original nine hardwares chosen in the initial stage without the Knuth Factor applied to their times. Figures 10-13 show the PDP 11/70, CDC 6600, CRAY 1, and HL-6/43 with the Knuth Factor applied to their applicable instructions. Figure 14 is a composite of eight of the original nine hardwares, without the Knuth Factor applied, shown on the same profile for comparison purposes. The IBM 360/30 was left off this composite, because the larger scale would have made the profiles difficult to see. Figures 15

and 16 are profiles of the hardwares shown on Figure 14, but separated into two graphs for better clarity. Figure 17 shows the composite profiles of the PDP 11/70, CDC 6600, CRAY 1, and HL-6/43 with the Knuth Factor applied. The six micro-computers chosen to exhibit the strength of the IMSET are shown in profile on Figure 18 through Figure 23. Figure 24 is the composite of five of the six micros. The INTEL 8080 was omitted from the composite for the same graphics scale reason as the IBM 360/30. Table VII provides a key for the instruction mixes listed by letter for each of the computer profiles.

4. Analysis of Execution Profiles

It is interesting to note that the Knuth Factor does indeed have an impact upon the sensitivity of the various hardwares to the various applications. On machines which use functional units to execute all of their instructions (CDC 6600, CRAY 1) the impact of the Knuth Factor is significant, while in machines which have only selected instructions enhanced (PDP 11/70, HL-6/43) the impact is significant only for certain applications.

When analyzing the profiles it is important to remember that the purpose of the IMSET is to compare a machine's execution time sensitivity between applications, not only its estimated effective execution speed for any one application. The sensitivity between applications is determined by comparing the times of execution as a percentage. Two examples are presented to illustrate the use of the IMSET.

The first example illustrates the sensitivities on the data obtained from the PDP 11/70 profile with the Knuth Factor accounted for, and the CRAY 1 profile with Knuth Factor accounted for. The second example provides data obtained from the micro-computer profiles. The example assumes a micro-computer selection to handle navigation and telemetry (NAVSAT receiver, for example) applications.

a. Profile Analysis Example 1

In this example, the sensitivites of the PDP 11/70 and the CRAY 1 will be compared using the execution times for the scientific, navigation, and real-time mixes.

MIX	PDP 11/70 Execution (μsec)	CRAY 1 Execution (μsec)
SCIENTIFIC	1.718	0.060
NAVIGATION	2.646	0.044
REALTIME	2.617	0.060

PDP 11/70 Sensitivities

	Sci.	Nav.	R-T
Sci	---	54% Faster	52% Faster
Nav	---	---	1% Slower
R-T	---	---	---

CRAY 1 Sensitivites

	Sci.	Nav.	R-T
Sci	---	36% Slower	0%
Nav	---	---	36% Faster
R-T	---	---	---

This abbreviated example shows that the CRAY 1 is less sensitive to the three mixes than is the PDP 11/70, because there is only a 36% difference between its execution speeds over the three mixes as opposed to the PDP 11/70's

difference of 54% maximum sensitivity. Assuming only the broad area for future use of a hardware were known (scientific, navigation, or some type of real-time application) this example points out that the CRAY 1 would best fit the application, because its sensitivity to the areas of suspected applications is much less than that of the PDP 11/70's.

When used in actual practice the sensitivity matrix will grow much larger as more mix applications are accounted for. Each pair of mixes being compared need be done only once, because if Mix A executes 35% faster than Mix B, then Mix B is also 35% slower in execution than Mix A. With the sensitivities available for all the machines to be evaluated the decision maker is then able to select the appropriate hardware based upon the machine exhibiting the lease sensitivity to the intended applications.

b. Profile Analysis Example 2

In this example a micro-computer is to be selected to handle both navigation and telemetry applications. The micro-computer selected should present the smallest change between the two applications (since the eventual percentage of workload is not known). The Digital Equipment Corp LSI 11/23 and the Motorola 68000 will be used for the purpose of this example.

MICROS	NAV (μSEC)	TLM (μSEC)	SENSITIVITY (%)
MOTOROLA 6800	10.842	8.625	26%
DIGITAL EQUIP CORP LSI 11/23	11.314	6.168	84%

This analysis shows that the MOTOROLA 68000 might be the more preferable micro-computer due to its lower sensitivity (more uniform performance) to the difference between the two applications (i.e. better worst case performance).

TABLE VII

KEY TO THE INSTRUCTION MIXES PRESENTED
IN FIGURE 1 THROUGH FIGURE 24

Letter	Instruction Mix
a	PROCESS CONTROL
b	MESSAGE PROCESSING
c	REAL TIME
d	COMMUNICATION CONTROL
e	DATA COMPRESSION
f	NAVIGATION
g	TLM THRUPUT
h	TECHNICAL GENERAL
i	SCIENTIFIC
j	COMPOSITE GENERAL

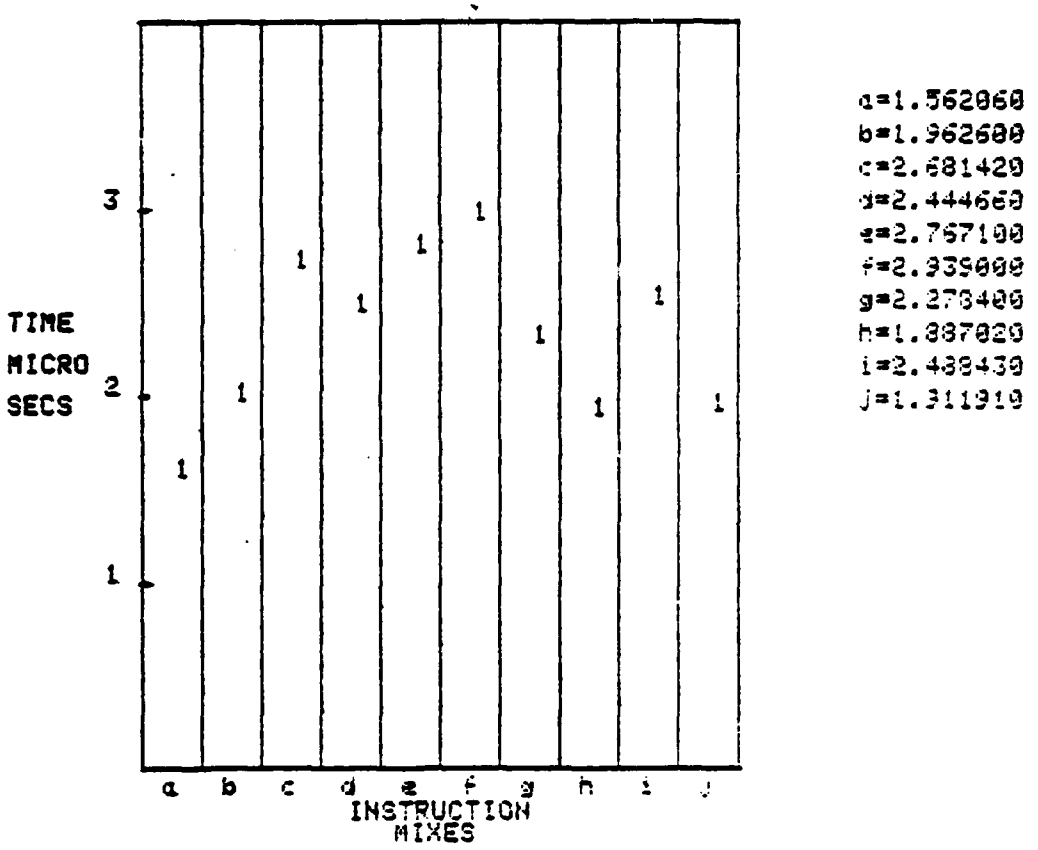


Figure 1. PDP 11/70 without Knuth Factor.

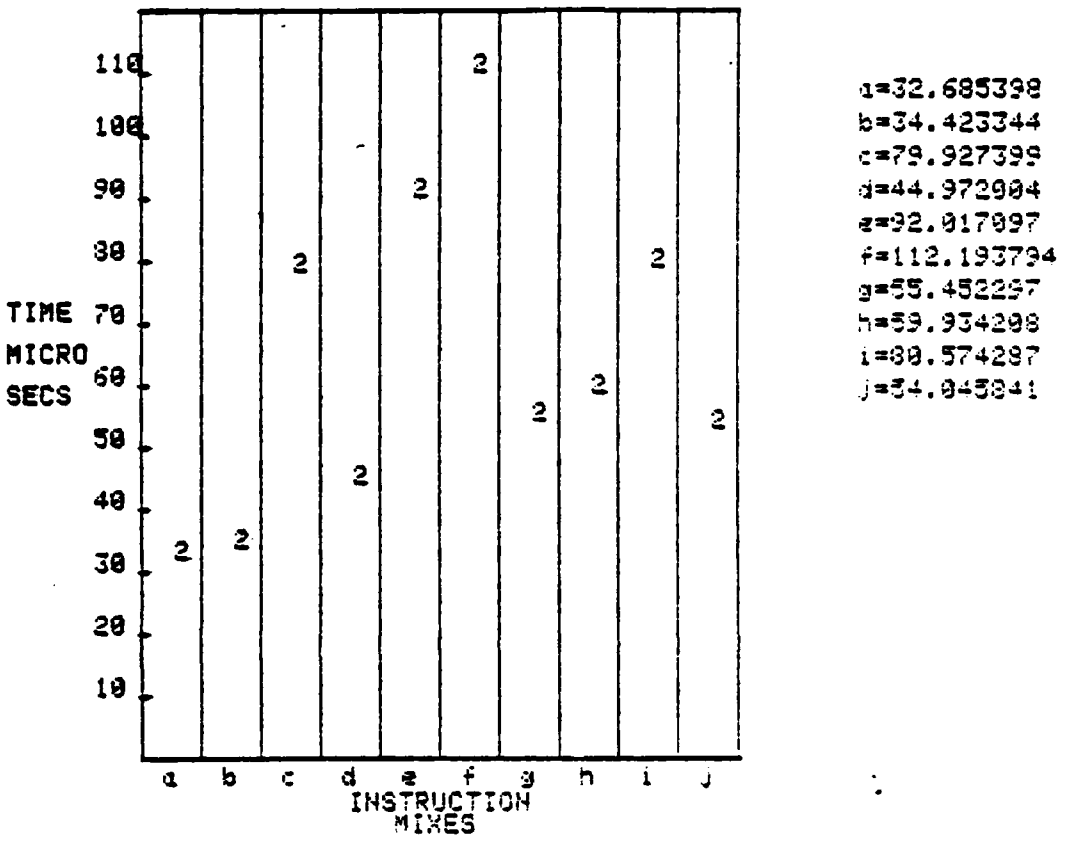


Figure 2. IBM 360/30

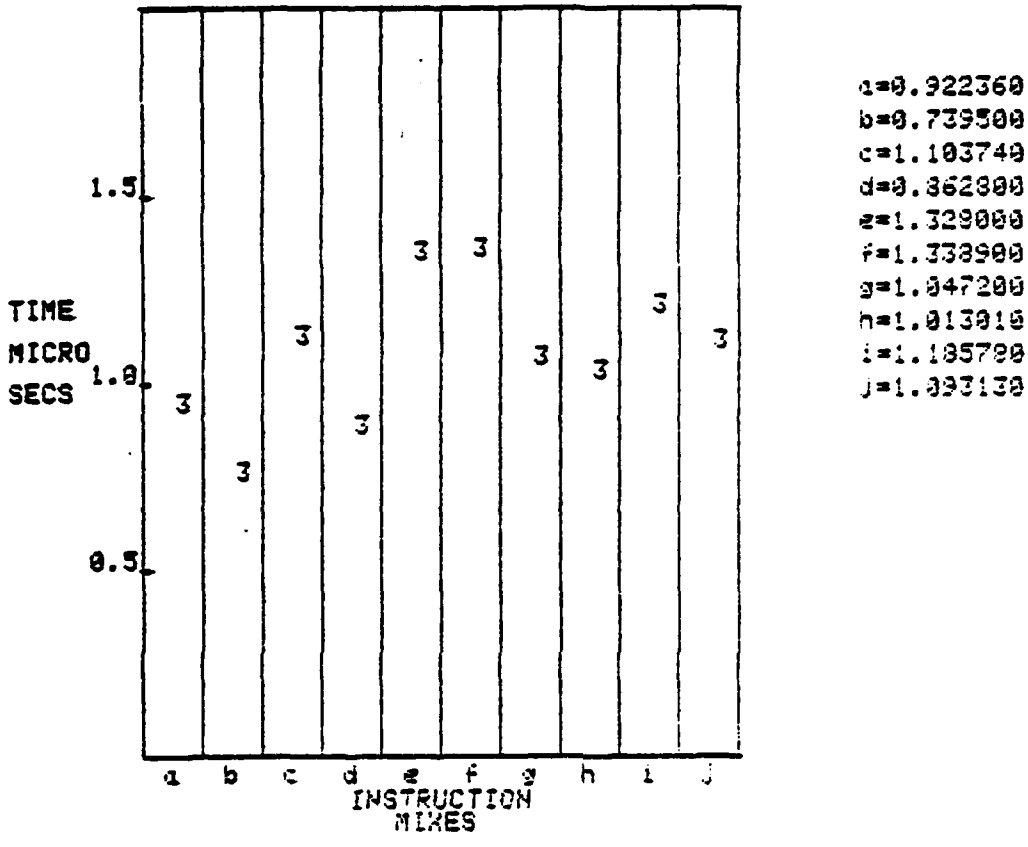


Figure 3. IBM 360/75

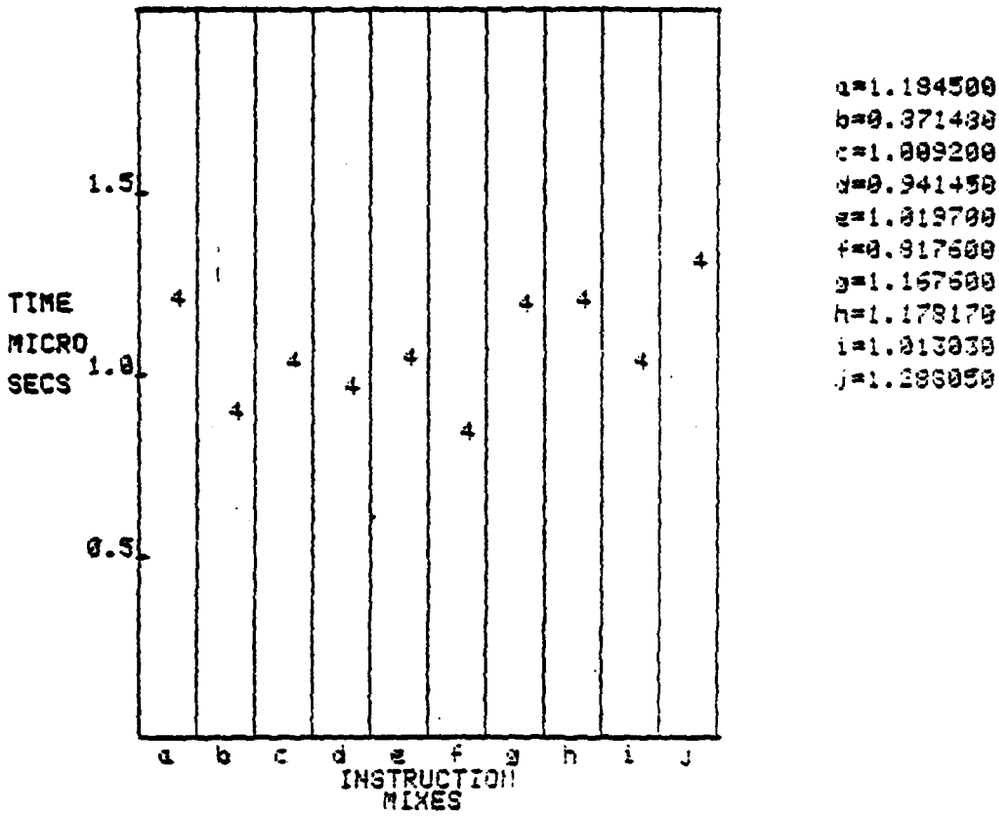


Figure 4. CDC 6600 without Knuth Factor.

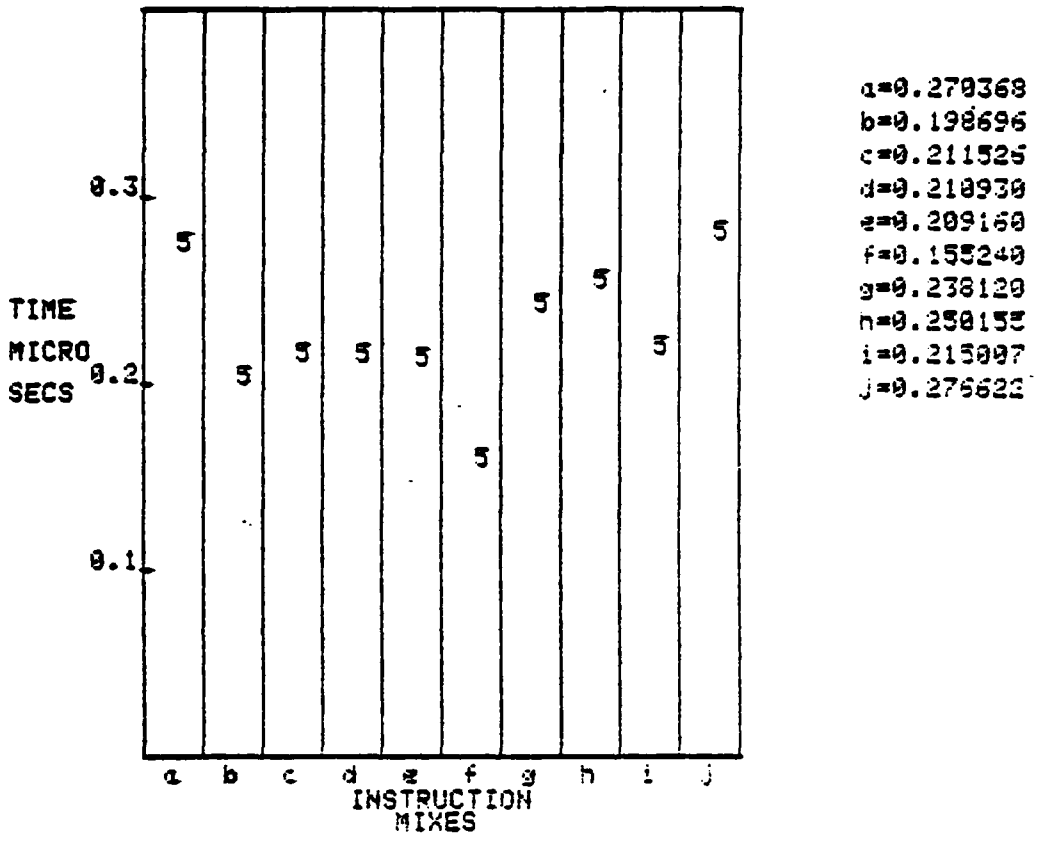


Figure 5. CRAY 1 without Knuth Factor.

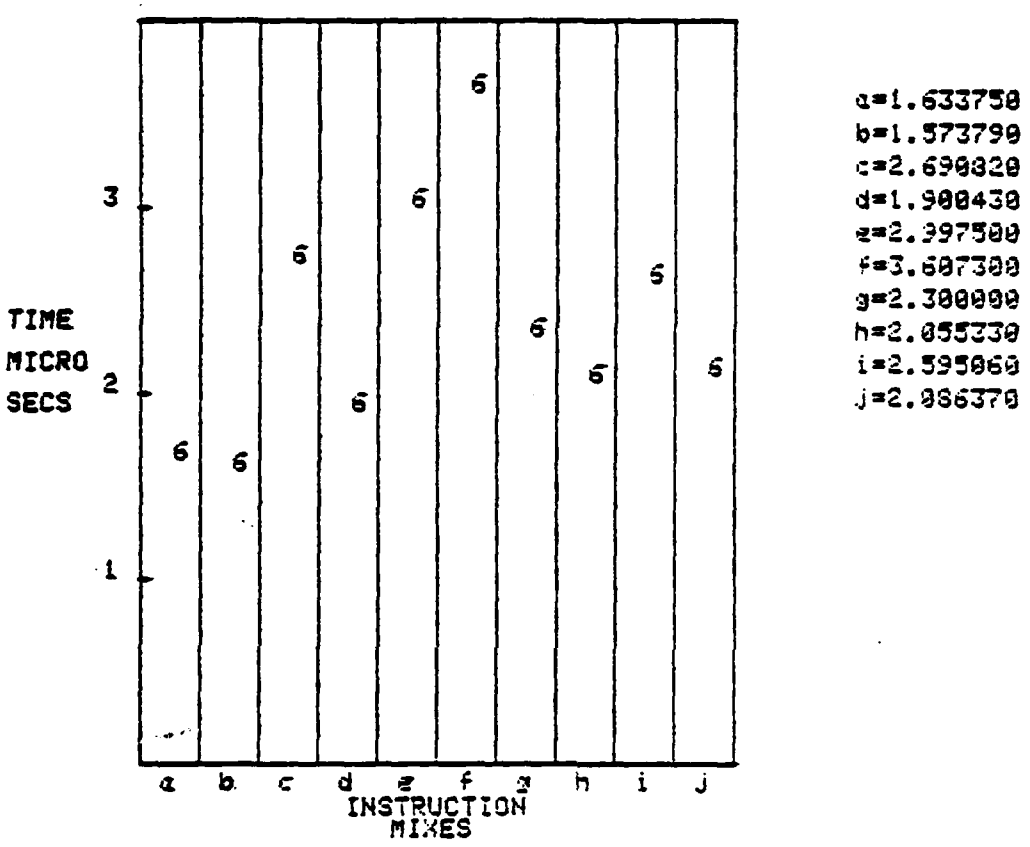


Figure 6. Honeywell Level-6/43 without Knuth Factor.

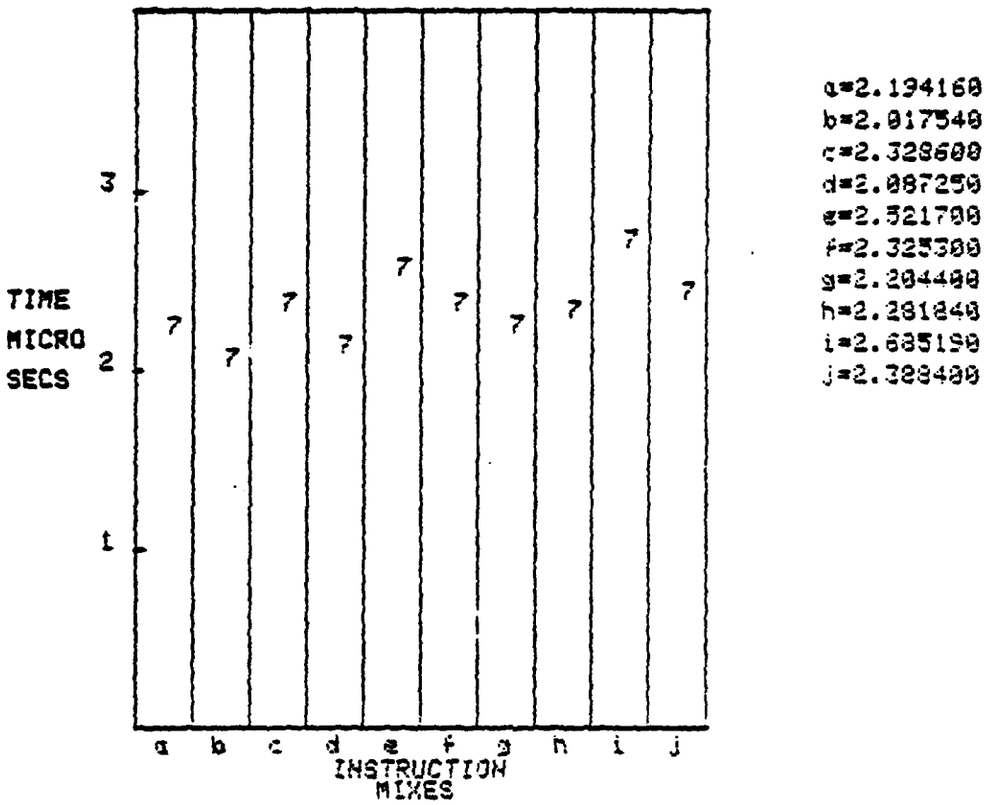


Figure 7. AN/UYK-20

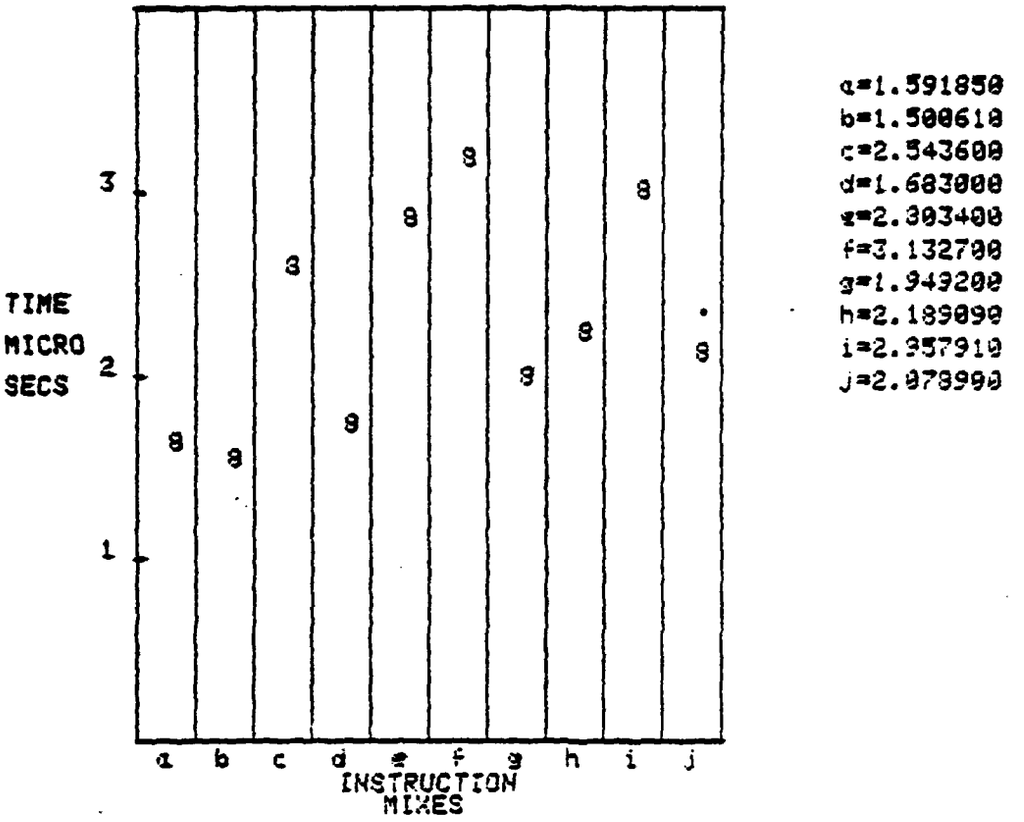


Figure 8. AN/UYK-7

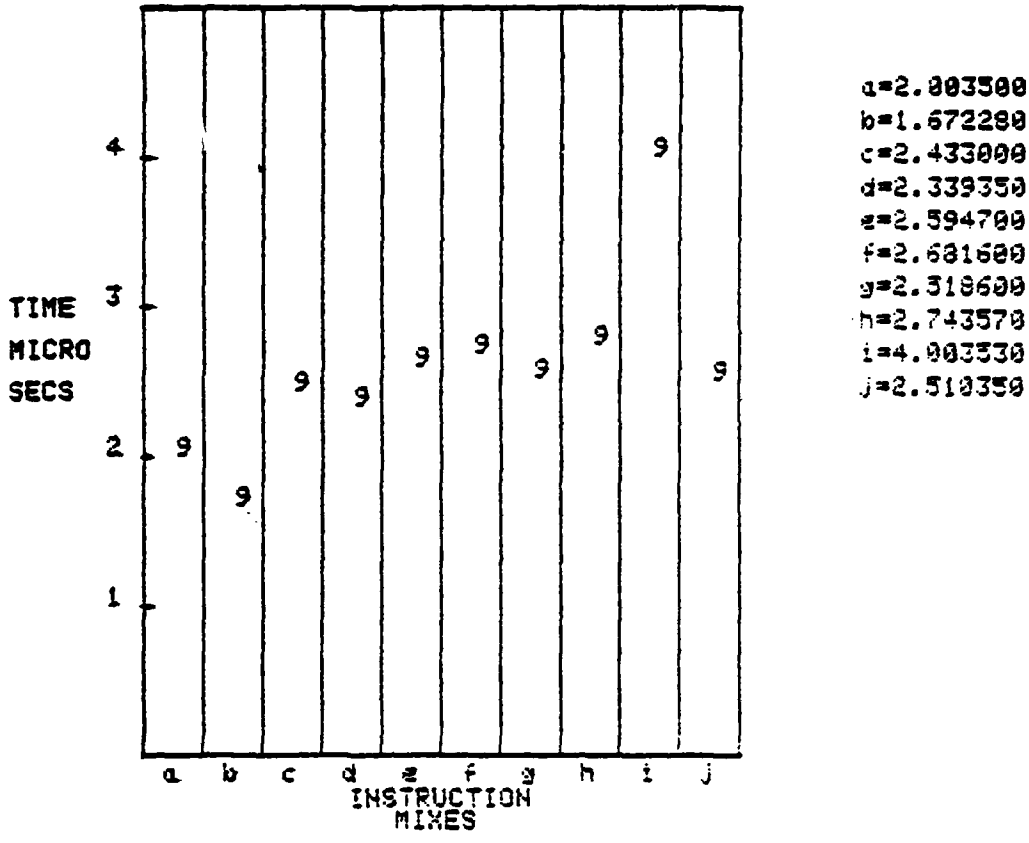


Figure 9. AN/AYK-14(V)

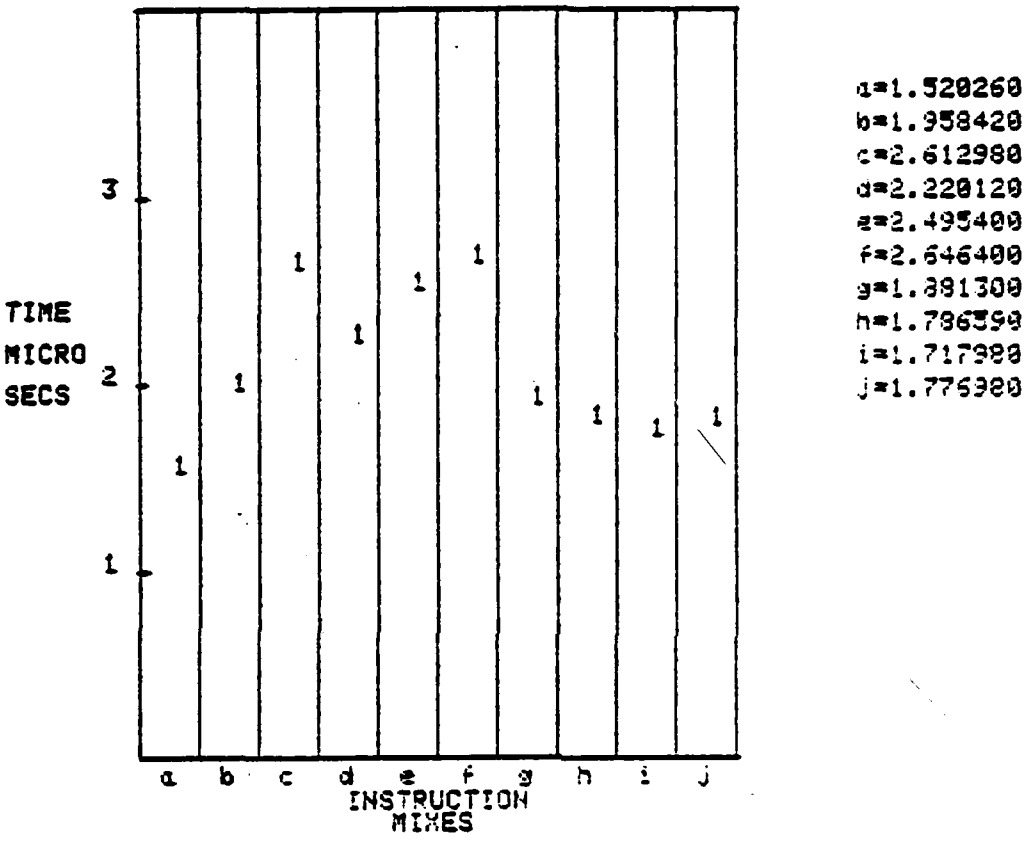


Figure 10. PDP 11/70 with Knuth Factor applied.

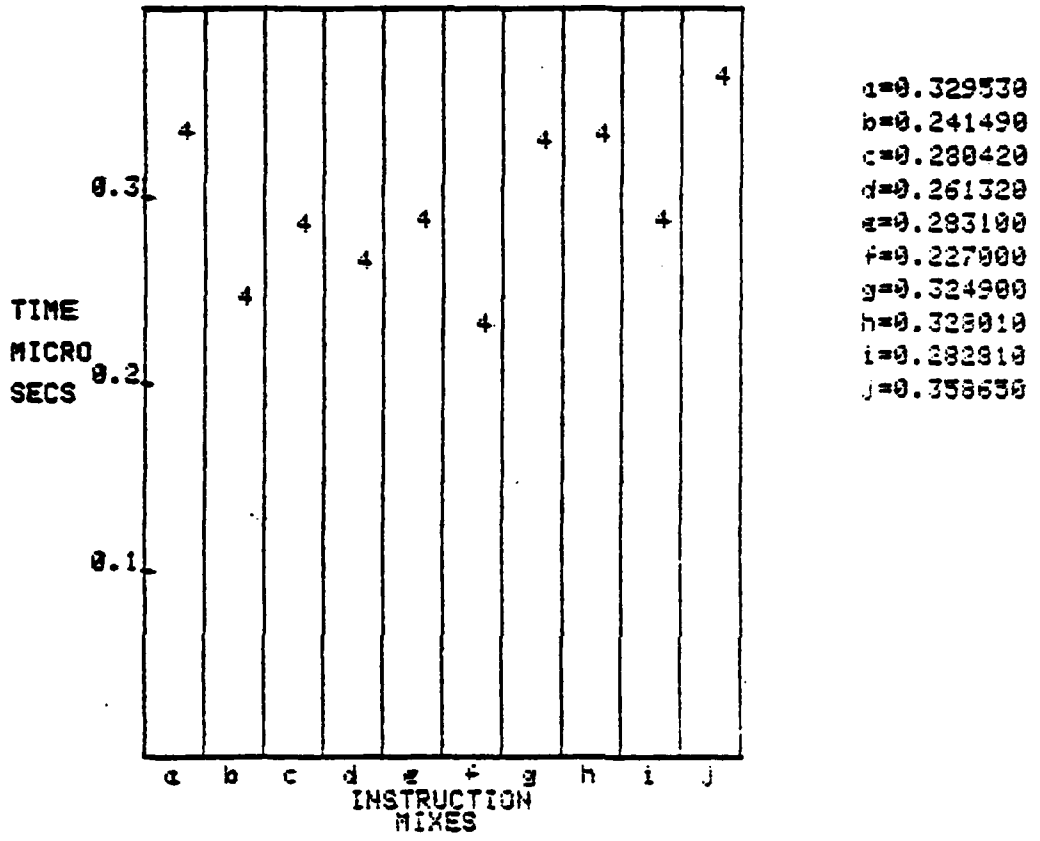


Figure 11. CDC 6600 with Knuth Factor applied.

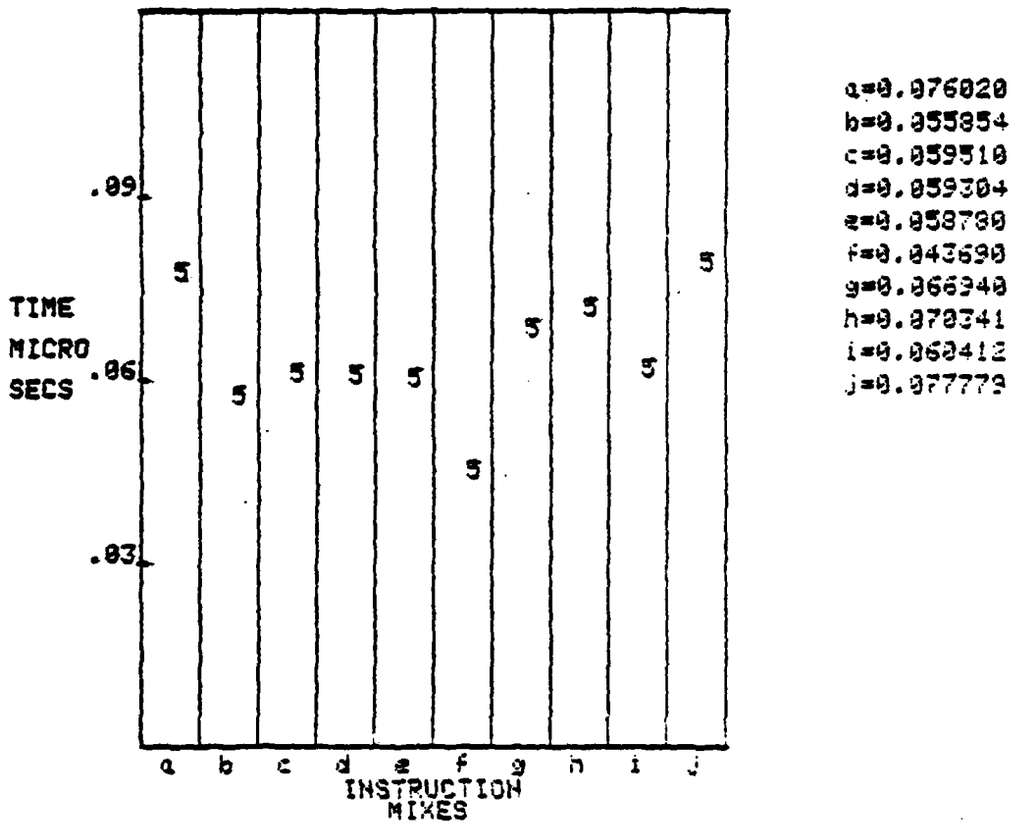


Figure 12. CRAY 1 with Knuth Factor applied.

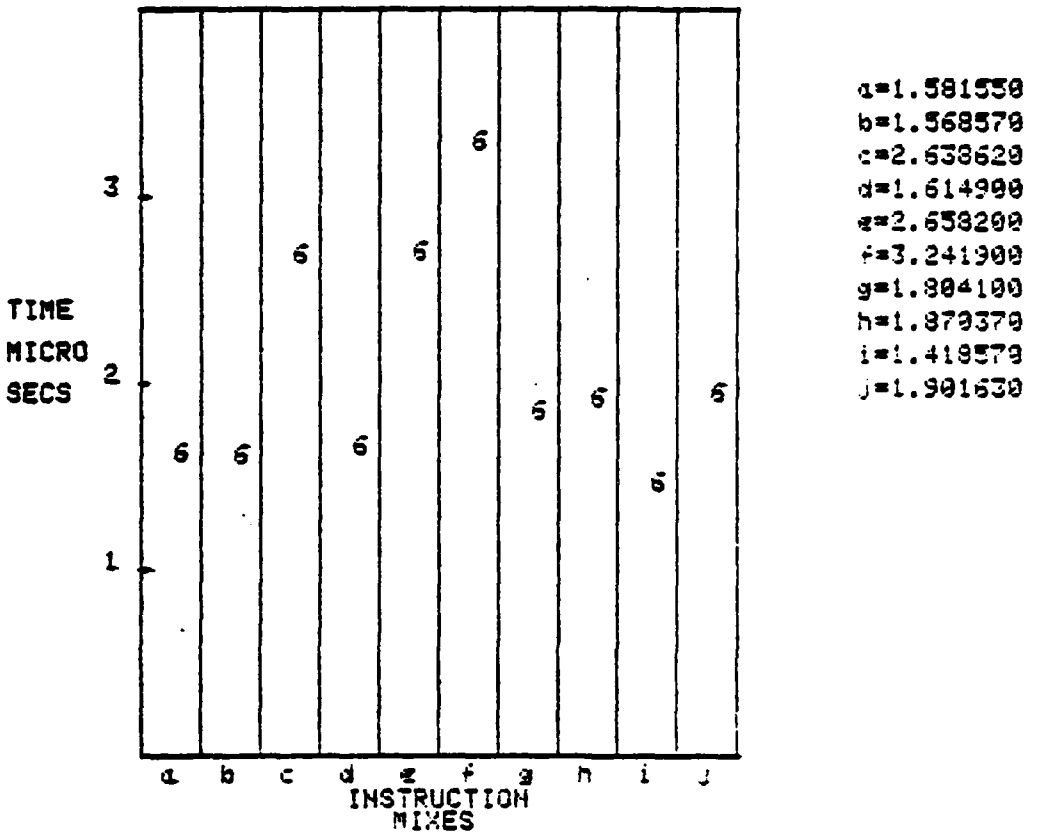


Figure 13. Honeywell Level-6/43 with Knuth Factor applied.

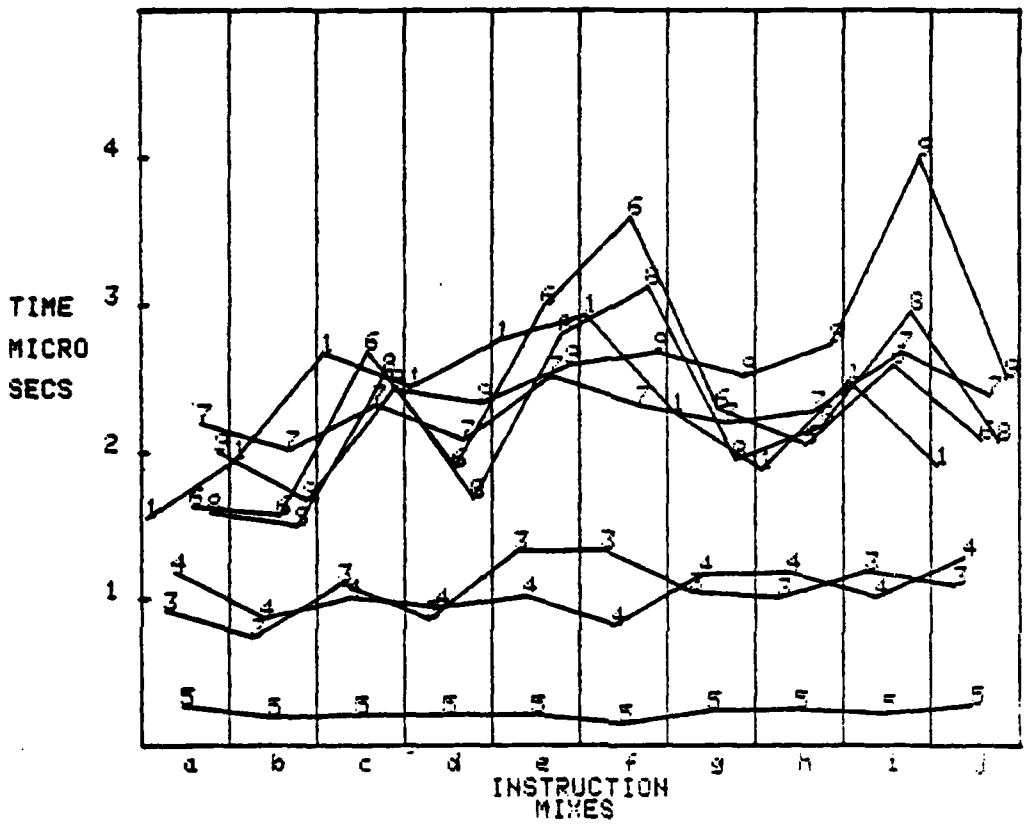


Figure 14. Composite sensitivity profiles without Knuth Factor applied: (1) PDP 11/70, (3) IBM 360/75, (4) CDC 6600, (5) CRAY 1, (6) HONEYWELL LEVEL-6/43, (7) AN/UYK-20, (8) AN/UYK-7, (9) AN/AYK-14(V).

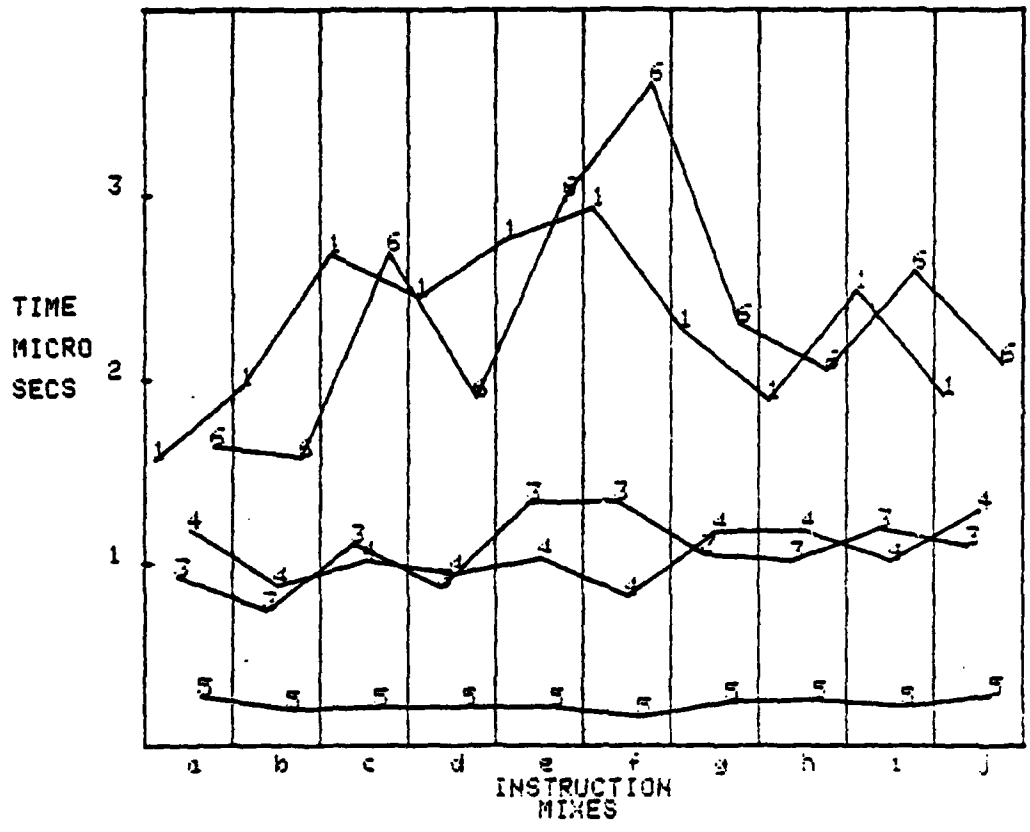


Figure 15. Composite sensitivity profiles without Knuth Factor applied: (1) PDP 11/70, (3) IBM 36-/75, (4) CDC 6600, (5) CRAY 1, (6) HONEYWELL LEVEL-6/43.

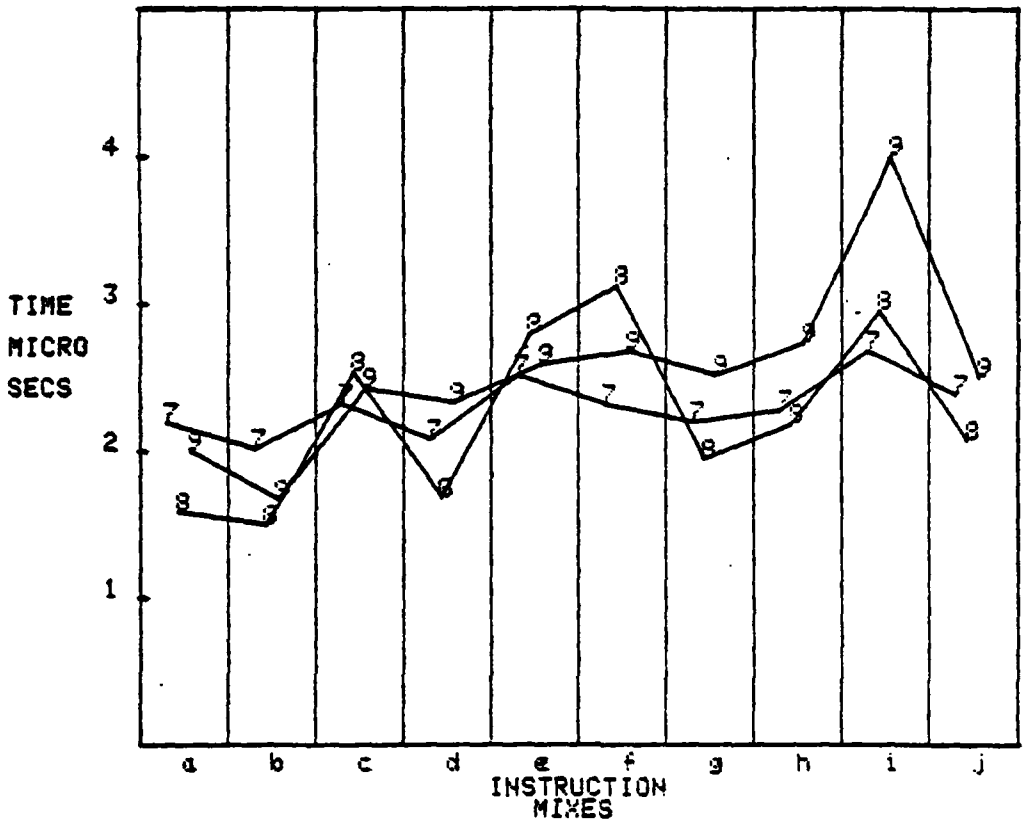


Figure 16. Composite sensitivity profiles without Knuth Factor applied: (7) AN/UYK-20, (8) AN/UYK-7, (9) AN/AYK-14(V).

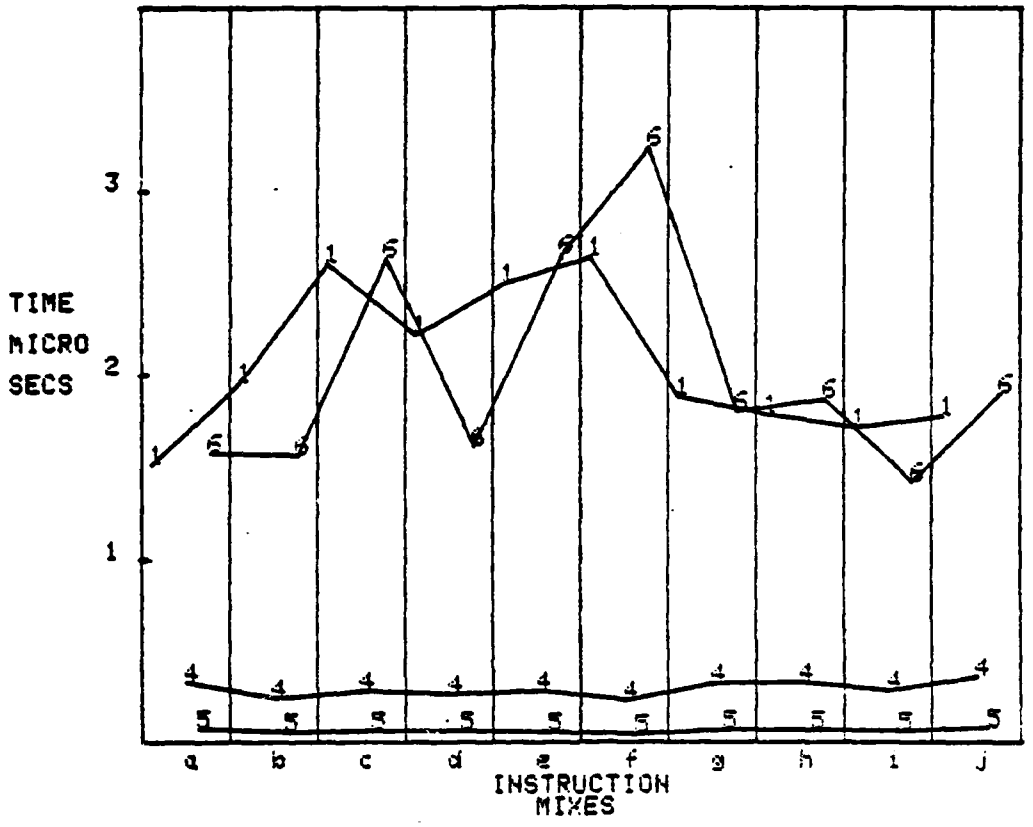


Figure 17. Composite sensitivity profiles with Knuth Factor applied: (1) PDP 11/70, (4) CDC 6600, (5) CRAY 1, (6) HONEYWELL LEVEL-6/43.

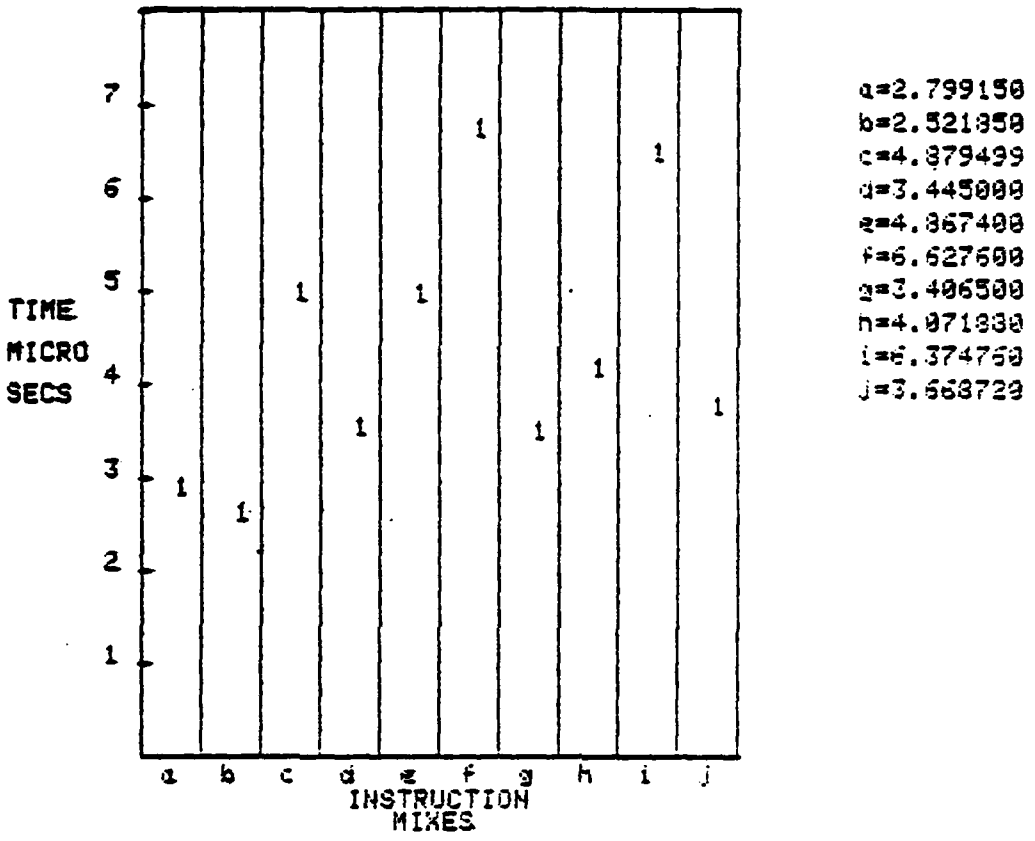


Figure 18. ZILOG 8000

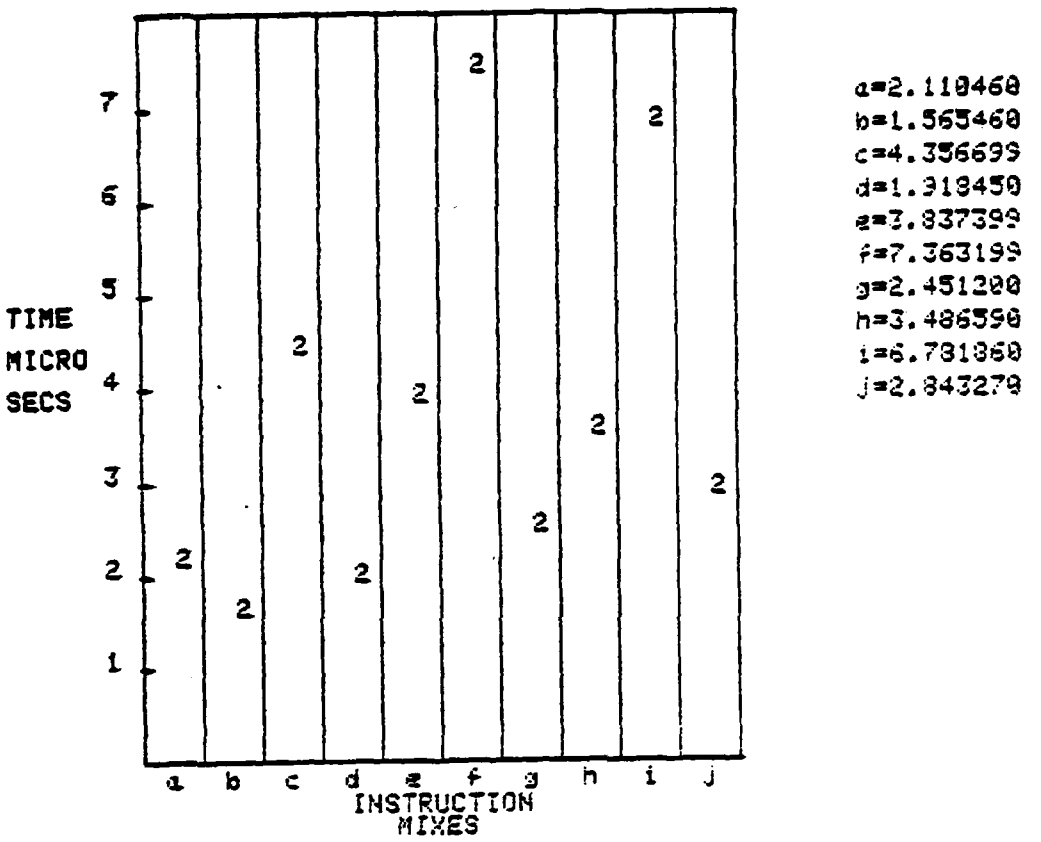


Figure 19. INTEL 8086

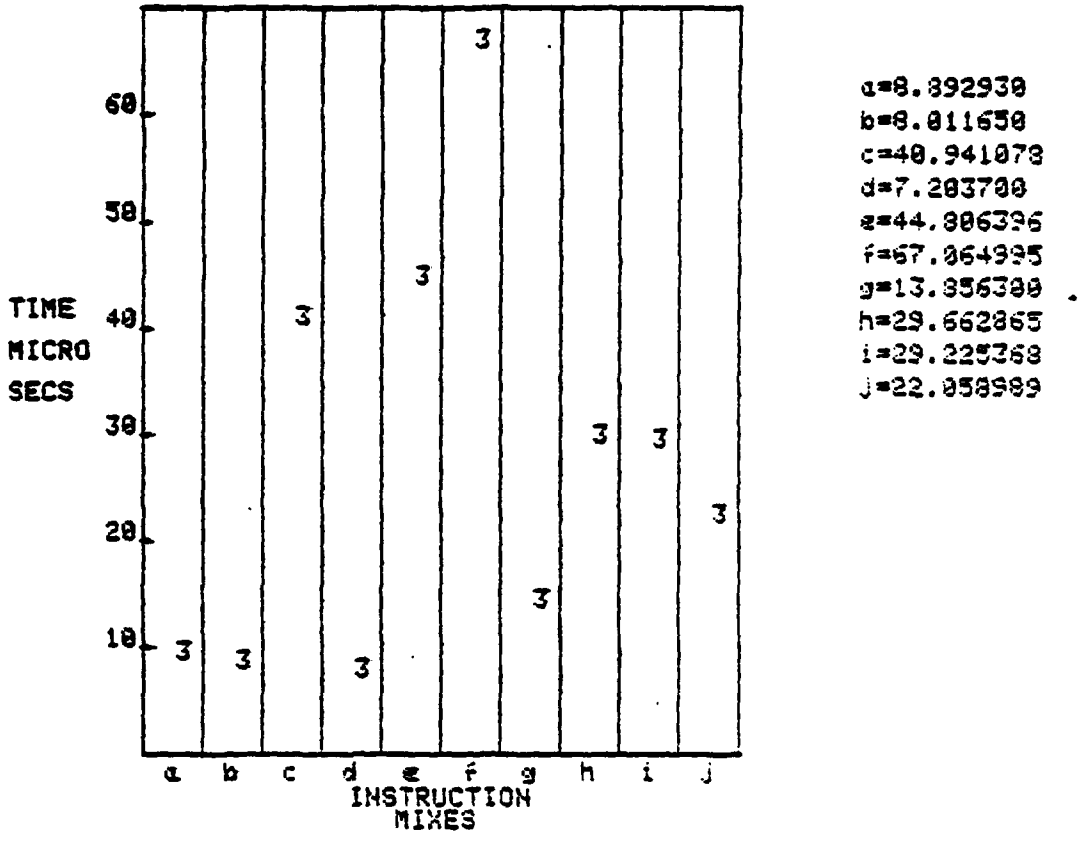


Figure 20. INTEL 8080

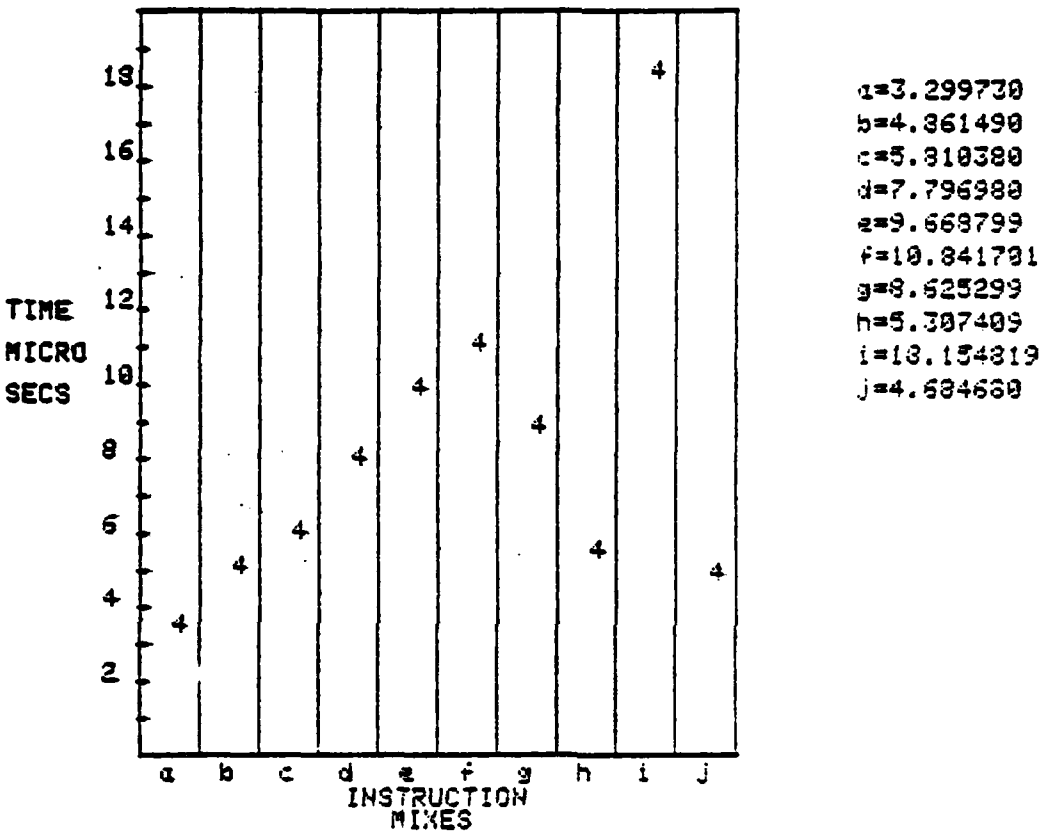


Figure 21. MOTOROLA 68000

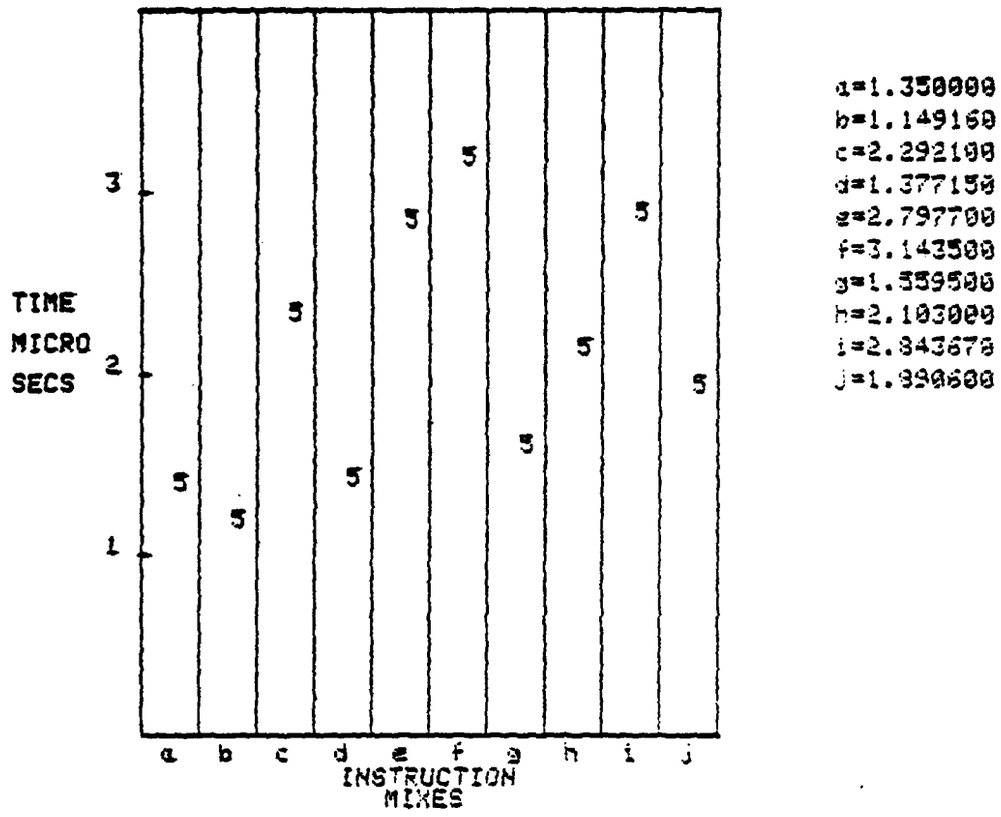


Figure 22. TEXAS INSTRUMENTS 9900

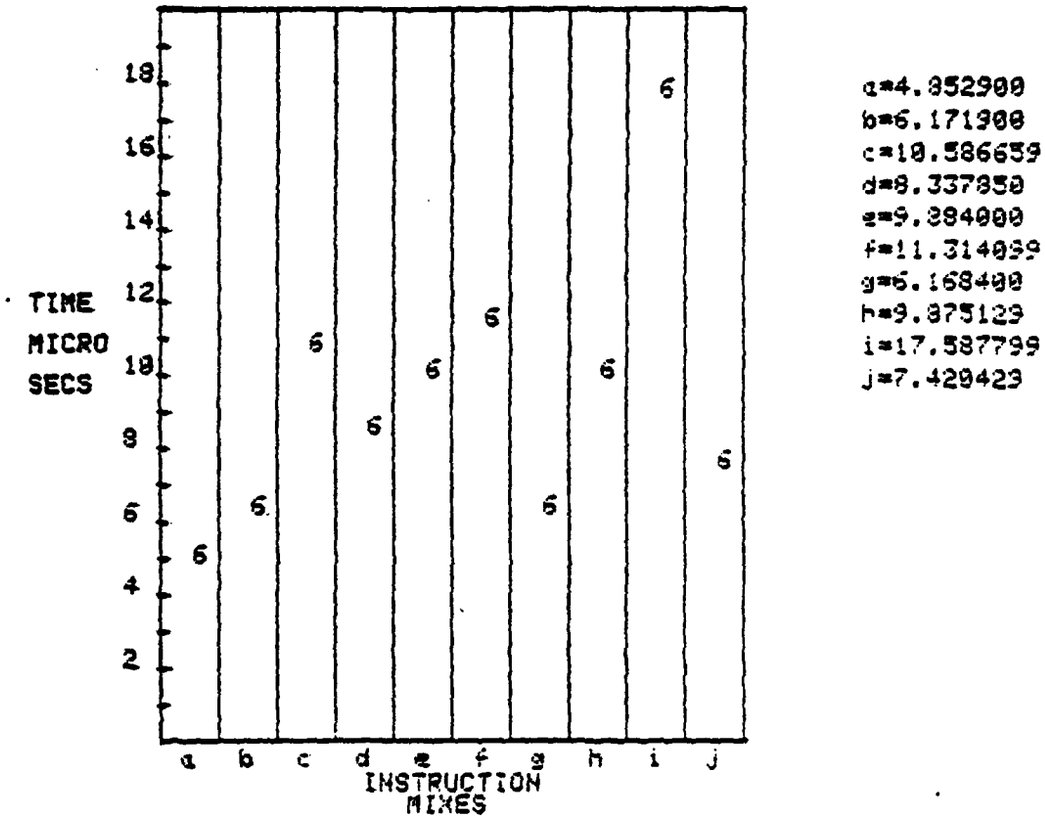


Figure 23. DIGITAL EQUIPMENT CORP LSI 11/23

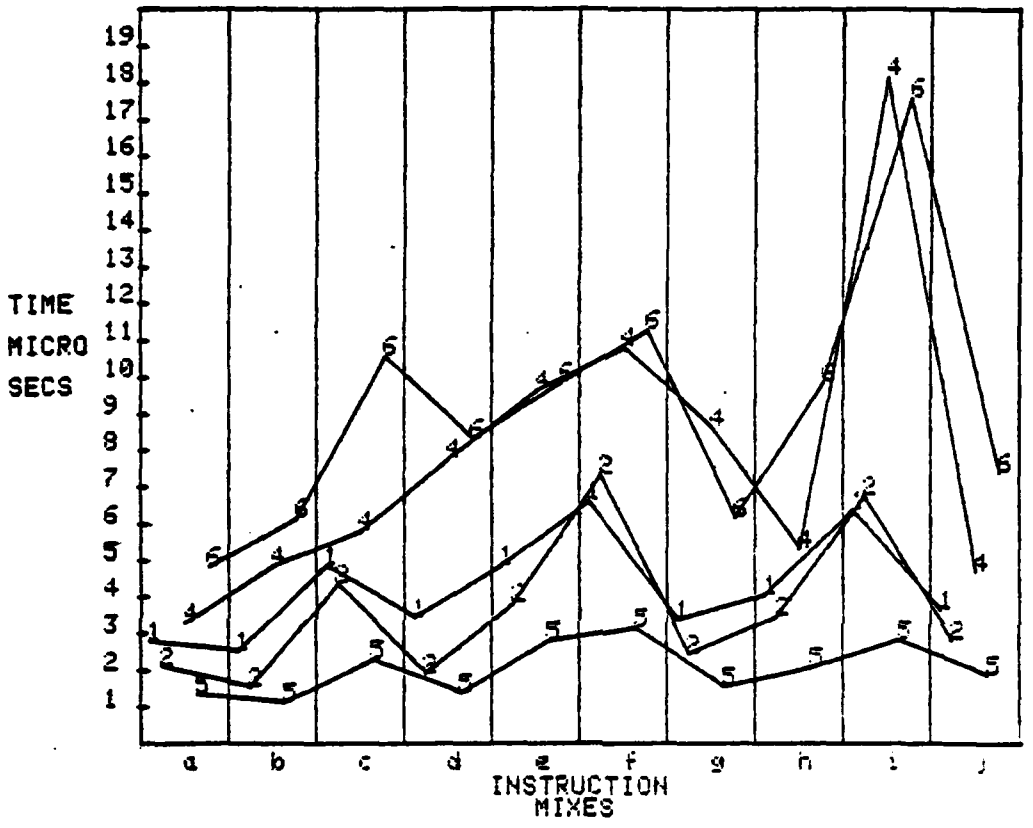


Figure 24. Composite sensitivity profiles of microcomputer hardwares: (1) ZILOG 8000, (2) INTEL 8086, (4) MOTOROLA 68000, (5) TEXAS INSTRUMENTS 9900, (6) LSI 11/23.

V. CONCLUSIONS

This thesis demonstrated a method, the IMSET, with which the government decision maker can quickly and efficiently select a computer hardware from a number of candidates. An example of how to apply the method was presented and profiles of actual hardwares were shown.

The application of the IMSET itself does not present a problem. Difficulties may arise when machine instruction execution times are being determined. A machine's instruction set may not contain an instruction needed to perform a particular IMSET function. The evaluator is faced with deciding what should be entered, which can be difficult and requires some time.

The strength of the IMSET as an evaluation tool lies in that fact that it is able to be applied in the absence of available hardware and specific knowledge of intended application. It is very important that a tool such as the IMSET be an integral part of any decision making process affecting the procurement of computer systems in the future.

APPENDIX A

INSTRUCTION MIXES

All mixes acquired during the development of the IMSET are provided in Table VIII. The first ten mixes comprise the IMSET. The next ten mixes, along with those comprising the IMSET, were used in the initial demonstration stage. The last two mixes were eliminated from the initial demonstration prior to evaluation due to lack of sufficient information.

The remainder of this Appendix section sets forth the references from which the mixes were acquired, and how the functional instruction weights were determined, if known.

1. MESSAGE PROCESSING

   Ref: [4]

   Comments: (Minimal information available concerning this mix's origin and development.)

2. PROCESS CONTROL

   Ref: [4]

   Comments: (Minimal information available concerning this mix's origin and development.)

3. COMMAND AND CONTROL

   Ref: [4]

   Comments: This mix was developed on the IBM 7090. It

is a compilation of actual instruction counts, and the author's experience in similar applications.

4. DATA COMPRESSION

Ref: [4]

Comments: (Minimal information available concerning this mix's origin and development.)

5. NAVIGATION

Ref: [4]

Comments: (Minimal information available concerning this mix's origin and development.)

6. TLM THRUPUT

Ref: [4]

Comments: (Minimal information available concerning this mix's origin and development.)

7. TECHNICAL/GENERAL

Ref: [6]

Comments: Developed on IBM 360. The weights were determined through the analysis of a library of trace programs. The mix is a combination of technical compiler (50%) and technical object (50%).

8. SCIENTIFIC

Ref: [5]

Comments: Developed on IBM 7000 series. Weights determined by a dynamic trace of a large number of scientific

and engineering applications. This mix typifies a general scientific area.

9. REAL-TIME

Ref: [9]

Comments: (Minimal information available concerning this mix's origin and development.)

10. GENERAL-COMPOSITE

Ref: [6]

Comments: Developed on the IBM 360. Weights determined through a library of trace programs. This mix is a combination of five types of programs: SORT (50%), COBOL-COMPILE (5%), COBOL-OBJECT (60%), TECHNICAL-COMPILE (15%), and TECHNICAL-OBJECT (15%).

11. GIBSON

Ref: [3]

Comments: Developed on IBM 704, and IBM 650. Weights determined by dynamic trace of predominately scientific jobs, approximately nine million instruction executions. Most well known of all instruction mixes developed to date.

12. COMMUNICATIONS

Ref: [10]

Comments: Developed from Honeywell 6000 series. Weights drawn from the examination of various communication software developed by Honeywell.

13. EDP

Ref: [4]

Comments: (Minimal information available concerning this mix's origin and development.)

14. RADAR DATA PROCESSING

Ref: [4]

Comments: (Minimal information available concerning this mix's origin and development.)

15. CONTROL AND DISPLAY

Ref: [4]

Comments: (Minimal information available concerning this mix's origin and development.)

16. COMMAND AND CONTROL

Ref: [4]

Comments: (Minimal information available concerning this mix's origin and development.)

17. TRACK AND COMMAND

Ref: [4]

Comments: (Minimal information available concerning this mix's origin and development.)

18. RADAR SEARCH AND TRACK

Ref: [4]

Comments: (Minimal information available concerning this mix's origin and development.)

19. REAL-TIME

Ref: [4]

Comments: (Minimal information available concerning this mix's origin and development.)

20. GENERAL PURPOSE

Ref: [4]

Comments: (Minimal information available concerning this mix's origin and development.)

21. COMMERCIAL

Ref: [7]

Comments: Developed on IBM 705. Weights determined from nine programs involving over one million operations. Programs included inventory, general accounting, billing, payroll, and production planning.

22. SCIENTIFIC

Ref: [7]

Comments: Developed on IBM 704, 7090. Weights determined from over 100 problems involving over 15,000,000 operations.

TABLE VIII

ORIGINAL TWENTY-TWO INSTRUCTION MIXES

	I. ARITHMETIC								II. LOGICAL			III. CONTROL						IV. I/O MISC.
	FIXED POINT (SP)			FIXED POINT (DP)		FLOATING POINT (SP)							BRANCH					
	ADD/SUB	MULTIPLY	DIVIDE	ADD/SUB	MULTIPLY	ADD/SUB	MULTIPLY	DIVIDE	COMPARE	SHIFT	AND/OR	LOAD/STORE	CONDITIONAL	UNCONDITIONAL	INC & STORE INDEX	MOVE	INDEX	I/O & MISC.
PROCESS CONTROL	.064	.013	.011	0.0*	0.0*	0.0*	0.0*	0.0*	0.0	.002	0.0	.400	.480	0.0*	0.0	0.0	0.0*	.020
MESSAGE PROCESSING	.050	.005	.005	0.0*	0.0*	0.0*	0.0*	0.0*	.010	.030	.150	.470	.140	0.0*	.030	.058	0.0*	.052
REAL TIME	.126	.108	.020	.014	.012	0.0*	0.0*	0.0*	.020	.070	.010	.500	.100	0.0*	0.0*	0.0*	0.0*	.020
COMMUNICATION/CNTL.	.080	.002	.002	0.0*	0.0*	.005	.001	.001	.075	.075	.070	.475	0.0*	.050	.030	0.0*	.035	.090
DATA COMPRESSION	.190	.060	.060	0.0*	0.0*	0.0*	0.0*	0.0*	.120	0.0	0.0	.270	.060	0.0*	.060	.060	0.0*	.120
NAVIGATION	.230	.250	0.0	0.0*	0.0*	0.0*	0.0*	0.0*	.020	0.0	0.0	.300	.020	0.0*	.040	0.0	0.0*	.140
TIM THRUPUT	.090	.040	0.0	0.0*	0.0*	0.0*	0.0*	0.0*	.290	0.0	0.0	.220	.150	0.0*	.040	0.0	0.0*	.190
TECHNICAL GENERAL	.025	.025	.025	0.0*	0.0*	.011	.011	.011	.108	.045	0.0*	.361	.276	0.0*	0.0*	0.0*	0.0*	.103
SCIENTIFIC	0.0*	0.0*	0.0*	0.0*	0.0*	.095	.050	.020	0.0*	0.0*	0.0*	.285	.132	0.0*	0.0*	0.0*	.225	.187
COMPOSITE GENERAL	.022	.022	.022	0.0*	0.0*	.003	.003	.003	.120	.029	0.0*	.361	.358	0.0*	0.0*	0.0*	0.0*	.057
E D P	.126	.023	.009	0.0*	0.0*	0.0*	0.0*	0.0*	.106	.071	.035	.337	.200	0.0*	.031	.017	0.0*	.045
RADAR DATA PROC.	.066	.018	.003	0.0*	0.0*	0.0*	0.0*	0.0*	.109	.150	.112	.177	.238	0.0*	.035	.090	0.0*	.002
CONTROL/DISPLAY	.080	.006	.002	0.0*	0.0*	0.0*	0.0*	0.0*	.007	.050	.060	.470	.220	0.0*	.090	.010	0.0*	.005
COMMUNICATION/CNTL.	.085	.005	.002	0.0*	0.0*	0.0*	0.0*	0.0*	.065	.075	.070	.475	.125	0.0*	.065	.010	0.0*	.023
TRACK & COMMAND	.180	.050	.020	0.0*	0.0*	0.0*	0.0*	0.0*	0.0	.070	.030	.380	.200	0.0*	0.0	.070	0.0*	0.0
RADAR SEARCH TRACK	.053	.013	.004	0.0*	0.0*	0.0*	0.0*	0.0*	.041	.074	.049	.485	.241	0.0*	.014	.005	0.0*	.021
GIBSON	.061	.006	.002	0.0*	0.0*	0.0*	0.0*	0.0*	.038	.044	.016	.312	.160	0.0*	.180	.053	0.0*	.050
REAL TIME	.160	.050	.020	0.0*	0.0*	0.0*	0.0*	0.0*	.120	.050	.040	.330	.100	0.0*	.040	.050	0.0*	.040
GENERAL PURPOSE	.107	.036	.012	0.0*	0.0*	0.0*	0.0*	0.0*	.086	.051	.045	.337	.184	0.0*	.048	.041	0.0*	.053
COMMUNICATIONS	.150	0.0*	0.0*	0.0*	0.0*	0.0*	0.0*	0.0*	0.0*	.050	.050	.400	.150	.150	0.0*	0.0*	0.0*	.050
KNIGHT COMMERCIAL	.250	0.0*	0.0*	0.0*	0.0*	0.0	.010	0.0	0.0*	0.0*	0.0*	0.0*	0.0*	0.0*	0.0*	0.0*	0.0*	0.0*
KNIGHT SCIENTIFIC	.100	0.0*	0.0*	0.0*	0.0*	.100	.060	.020	0.0*	0.0*	0.0*	0.0*	0.0*	0.0*	0.0*	0.0*	0.0*	0.0*

* Weight not assigned by mix for this functional instruction.

APPENDIX B

DEFINITION OF FUNCTIONAL INSTRUCTIONS

This appendix sets forth what is meant by each of the functional instructions, and in general, how each of the functional instruction execution times were calculated for a particular machine.

The functional instructions utilized in the IMSET were determined by combining the selected mixes. Those instructions representing basic operations were then chosen as the first seventeen instructions in the IMSET. The remaining instructions with their weights were combined under the eighteenth functional instruction, I/O & Miscellaneous.

Before preceeding to determine each machine's instruction execution times, a standardization of each of the functional instructions had to be set up so that the times being determined for each machine were being done based upon common assumptions. The assumptions upon which the execution times were determined are set forth below.

A. ARITHMETIC INSTRUCTIONS

All the arithmetic instructions were taken as register-to-register operations. This was done so as to avoid the difficulty of having to account for the number of operands per instruction.

1. Substitute Time Determination

Few machines possess, as part of their instruction sets, all the arithmetic instructions listed as part of the IMSET. For example, the microprocessors, with the exception of the LSI 11/23, do not include floating point operations. When this type of situation arose a suitable time had to be calculated by an alternate method. Simply entering a time of zero for missing instructions was not acceptable, because a machine with few instructions would appear to execute faster than a machine with a powerful instruction set. Penalty times to compensate for missing instructions were determined by three methods. The first method involved an acceptable algorithm using available instructions from a machine's instruction set to accomplish the required operation. The summation of the instruction times included in the algorithm were then entered as the time required to execute the missing operation. The second method involved a knowledge of how a hardware executes a particular operation. This was the method used to determine the floating point execution times for the hardwares which do not have those instructions. A floating point operation, for instance multiply, generally involves a fixed point ADD of the exponents, a fixed point MULT of the mantissas, and a number of shifts for normalizations. The execution times for these fixed point operations are totaled, and the result is entered into the appropriate functional floating point instruction as the execution time.

It should be noted that applying this method of compensation to the LSI 11/23, which has floating point instructions, preserves its ranking in relation to the other micro-processors presented here. The execution times calculated by the compensation method for the LSI 11/23 range from approximately 11 microsecs faster for a floating point division to almost 35 microsecs faster for a floating point multiplication. The LSI 11/23 ranked sixth overall for floating point execution times using both the manufacturer's given execution times and the recalculated times using the compensation method. This would seem to indicate that even though the substitute times are not totally accurate they do provide an acceptable alternative when no times are available.

The last method used to determine a substitute execution time involved simply entering a floating point operation execution time for the appropriate fixed point execution time. This penalty was felt to be reasonable based on the facts that floating point times are generally greater than the fixed point executions, and that if a particular machine was required to do a fixed point operation and that instruction was not a part of the instruction set then a floating point execution would be submitted.

B. LOGICAL INSTRUCTIONS

1. Compare

The compare instruction for the maxi-computers, and mini-computers were taken as register-to-memory operations.

For the micro-computers a compare was considered to be a register immediate operation, because it is the operation most common in a micro-computer's instruction set. Any deviations from this procedure is so indicated in the tables of execution times for each of the hardwares presented.

2. Shifts

For the maxi-computers and mini-computers a shift is considered to be an eight bit shift. For some of the hardwares presented the number of bits shifted does not make a difference (i.e. CRAY-1); while for others a shift involves a constant time plus some value times the number of bits shifted (i.e. PDP 11/70). A six bit shift was taken as the standard for the micro-computer.

3. And/Or

As with the arithmetic instructions all AND/OR operations were taken to mean register-to-register. The only exception to this standard was the TI-9900 which only utilizes immediate AND/OR instructions.

C. CONTROL INSTRUCTIONS

1. Load/Store

The load and store operation times presented another minor problem. Some hardwares provide no true load or store operations, but perform the function indirectly as a MOVE or as a READ or WRITE operation. The actual loading and storing timing information is contained in the other instructions as fetches from memory and returns to memory. The standard

chosen for this functional instruction was determined to be the time required for the processor to retrieve data from memory and place it into a working register or the time required to place data into memory from a working register. If a hardware's instruction set included LOAD and STORE instructions then the times indicated were used, otherwise a MOV register-to-memory instruction was chosen to be appropriate. Often the times required for the load and store operations were different. In all cases the average between the two times was used as the execution time of the LOAD/STORE operation.

2. Branch

The conditional branch execution times were determined by averaging all the branch instruction execution times except the unconditional case. In many instruction sets the times required for conditional branches varied depending upon whether the branch was taken or not taken, and whether the branch was to an instruction in main memory or in a cache memory. The time determined for each of the conditional branch instructions was worst case. For an unconditional branch if there was a difference in execution times between in stack or out of stack branch the worst case time was used.

3. Increment and Store Index

The sense of this functional instruction was to be able to increment a register and then store the value in memory as an index. For virtually all the hardwares evaluated

this operation had to be accomplished by means of more than one instruction. Normally an increment or an add instruction used with a store or move to memory instruction would accomplish this task. The execution time was then determined by totaling the times required to accomplish the operations.

4. Move

A move was determined to be the time required to move a word from one register to another register. There were no real problems with this functional instruction, because almost all hardwares incorporate register-to-register moves in their instruction sets.

5. Index

This instruction is the time required to accomplish an indexing through memory or through a register stack by means of index registers for a task such as vector addition. Not all machines incorporate an indexing function directly with one instruction. Those that do not have an index instruction with index registers, or an indexing mode of operation must use an alternate method to accomplish the task. The method used in this thesis was a small loop consisting of an increment or add immediate instruction. For future evaluation this should not be a problem, because the machines being developed today have either an index instruction, index registers, or an indexing mode of operation.

D. I/O & MISCELLANEOUS INSTRUCTIONS

1. I/O & Misc.

This functional instruction encompasses all the instructions of a particular hardware's instruction set that were not utilized in the initial seventeen functional instructions. All the unused instructions execution times were totaled, and divided by the total number not used. This was the execution time entered for this functional instruction class.

APPENDIX C

DETERMINATION OF EACH MACHINE'S INSTRUCTION TIMES

The instruction times for each computer presented are calculated according to the guidelines set forth in Appendix B. This Appendix will identify each computer evaluated, and indicate exactly which instructions and times were used to determine the execution time for each instruction of the sensitivity technique. All times indicated were obtained from manufacturer's specifications as presented in referenced hardware manuals and literature.

The computers presented in Tables IX.a through IX.i are the hardwares used in the initial demonstration evaluation. Tables IX.j through IX.o present the micro-computers evaluated in the final demonstration.

A. DEC PDP 11/70

Table IX.a

The PDP 11/70's execution times [14] are dependent on the instruction itself, the modes of addressing used, and the type of memory referenced. In the general case the instruction times are determined by:

INSTR. TIME = SRC + DST + EF

where, SRC time was determined by averaging the times for all modes, and DST time was determined in the same manner. The average was used, because an instruction could be issued

in any mode so by averaging all cases would be considered to some extent. The EF time was chosen directly from the manufacturer's handbook. All times are typical processor timing with core memory, and may vary +15% to -10%.

Double operand instructions are determined by the general case formula, with the exception of the MOV instruction,

MOV INST. TIME = SRC + EF.

Single operand instructions are determined by,

INST. TIME = DST + EF or INST. TIME = SRC + EF

depending upon which instruction is used.

Branch instructions are simply,

INST. TIME = EF.

To increase the effective execution speed, the 11/70 utilizes a 1,024 word cache memory. This reduces the time required for the CPU to fetch (READ) an instruction from memory. This is accounted for by a factor determined by the average number of times, called a READ HIT RATE, or P_h, cache memory. Read hits average 80-95% of all machine cycles with a P_h=90% considered to be typical. The following formula determines the additional time to be added to each instruction execution time:

$$1.02x(1-P_h) \text{ x (number of read cycles).}$$

The number of read cycles for each instruction was determined by averaging all read cycles for all modes. For SRC and DST the average number of read cycles is 1.5.

1. Floating Point Processor (FPP) FP11-C

In order to increase execution speed of certain instructions included in the 11/70's instruction set, a FPP has been installed as a separate unit. The FPP executes in hardware floating point instructions which previously were executed in software. The FP11-C greatly enhances machine execution times for applicable instructions. The FPP operates in parallel with the main processor. This parallelism, or overlap, is the special feature of a machine for which the Knuth Factor, developed in Appendix D, will account. The determination of the floating point instruction execution times utilizing the FP11-C are determined as follows:

Effective Execution Time (EF)=

	Load Class	Store Class
Preinteraction	450 nsec	450 nsec
+Address Calculation	488 nsec	488 nsec
+Wait Time	492 nsec	2972 nsec
+Resync Time	450 nsec	450 nsec
+Interaction	300 nsec	300 nsec
+Argument Transfer	600 nsec	600 nsec
+Disengage & Fetch	300 nsec	300 nsec
Total:	3080 nsec	5560 nsec

Preinteraction Time: constant 450 nsec.

Address Calculation Time: determined to be 484 nsec by taking average of all modes floating point instructions.

Wait Time: 492 nsec for LOAD CLASS instruction, 2972 nsec

for STORE CLASS instructions. Calculations are shown below.

Resync Time: If wait time 0, then 450 nsec; else 0 nsec.

Interaction Time: constant 300 nsec.

Argument Transfer: 300 nsec x (number of 16-bit words from memory) using two 16-bit words for calculation.

Disengage & Fetch Time: constant 300 nsec.

Wait Time =

Load Class Instructions:

F.P. Execution Time (Previous F.P. Instr.)	2480 nsec
-Disengage & Fetch (Previous Instr.)	-300 nsec
-CPU Execution Time for Interposing Non-Floating Point Instruction	-750 nsec
-Preinteraction Time	-450 nsec
-Address Calculation Time	-488 nsec
Average Wait Time =	492 nsec

Store Class Instructions:

F.P. Execution Time (Previous F.P. Instr.)	2480 nsec
-CPU Execution Time for Interposing Non-Floating Point Instruction	-750 nsec
-Disengage & Fetch (Previous Instr.)	-300 nsec
-Preinteraction Time	-450 nsec
If 0, then total = 0) Total:	980 nsec
+Floating Point Execution Time	2480 nsec
-Address Calculation Time	-488 nsec
Average Wait Time:	2972 nsec

F.P. Execution Time (Previous F.P. Instr.): determined to be 2480 nsec by averaging all floating point instruction worst case times.

CPU Execution Time for Interposing Non-Floating Point Instruction: The time shown, 750 nsec, is the execution time for the SOB instruction in the CPU instruction set.

The FPP instruction set utilizes two types of instructions, LOAD CLASS, and STORE CLASS. Each type are identified as such in the instruction set.

The wait time is the time that the CPU spends waiting for completion by the FPP of a previous floating point instruction in the case of the LOAD CLASS instruction. For STORE CLASS, wait time is the summation of the time during which the FPP completes a previous floating point instruction, and FPP execution time for the individual STORE CLASS instruction.

The Knuth Factor was applied to the instructions which would be executed by the FPP.

B. IBM 360/30

Table IX.b

The IBM 360/30 execution times [15] were determined without benefit of any special feature execution enhancement. All operations were determined to be register-to-register where feasible. Penalty times were assigned to the arithmetic operations which have no direct instruction. Those are fixed point (SP) MULT, DIV, and fixed point (DP) ADD/SUB, and

MUL. The penalties assigned were those times indicated for the corresponding floating point (SP) operations. The Knuth Factor was not applied to any of the instruction execution times of the IBM 360/30.

C. IBM 360/75

Table IX.c

The IBM 360/75 execution times [15] were determined without benefit of any special feature execution enhancement. All operations were determined to be register-to-register where feasible. Penalty times were assigned to the arithmetic operations which have no direct instruction. Those are fixed point (SP) MULT, DIV, and fixed point (DP) ADD/SUB, and MUL. The panalties assigned were those times indicated for the corresponding floating point (SP) operations. The Knuth Factor was not applied to any of the instruction execution times of the IBM 360/75.

D. CDC 6600

Table IX.d

The CDC 6600 instruction times are given in machine minor cycles [16]. A minor cycle is 100 nsec. All times are counted from the point when a functional unit has both input operands to when the instruction result is available in the specified result register. There are ten functional units in the 6600 which receive appropriate instructions routed from the CPU. The functional units are Branch (1), Boolean (1), Shift (1),

Add (1), Multiply (2), Divide (1), Fixed Add (1), and Increment (2). If a functional unit is not currently in execution the instruction is issued, otherwise the CPU holds the instruction until the unit is free. The Knuth Factor was applied to all the instruction execution times determined. The resulting execution times would result with optimal use of the functional units where the CPU would not have to wait for a unit to be free.

E. CRAY 1

Table IX.e

The CRAY 1 utilizes 12 functional units for instruction execution [17]. This feature allows for maximum overlapping of all instructions. Another execution enhancement utilized by the CRAY 1 is block transfers of instructions and data from memory into four instruction buffers. This feature reduces execution times by eliminating numerous memory references.

The CRAY 1 does not provide double precision instructions, although double precision computations with 95-bit accuracy is available through software provided by CRAY Reserach. In order to provide a reasonable time figure for double precision instructions in the demonstration, the times for floating point executions were used. This appears to be a reasonable penalty time in view of the fact that floating point operations are similar to the fixed point double precision operations when determining execution times.

The CRAY 1 does not utilize a direct divide instruction. Divide is accomplished in floating point format by use of a multiple instruction sequence utilizing reciprocal approximation. A fixed point divide operation is accomplished through a software algorithm using floating point hardware.

All times indicated for the CRAY 1 execution speeds were calculated assuming there were no hold-issue conditions involving the desired functional units availibility, and all register and buffers were always ready to accept the next instruction. The worst case times were taken when they were indicated as such, otherwise average times were used.

All instructions in the CRAY 1 instruction set are susceptible to overlapping so the Knuth Factor was applied to all execution times.

F. HONEYWELL LEVEL-6/43

Table IX.f

The execution times for the HL-6/43 were determined using the maximum times indicated for each instruction [18]. This assumes that the prefetch buffers are always empty, and a memory block transfer must be made. All times are for register addressing (SAF mode) utilizing a double-fetch EDAC memory.

Instruction execution enhancement exists with the addition of a Scientific Instruction Processor (SIP) for floating point and fixed point instructions. All operands in the SIP are in floating point format, and the fixed point operations are

AD-A096 302 NAVAL POSTGRADUATE SCHOOL MONTEREY CA F/G 9/2
A COMPUTER EVALUATION TECHNIQUE FOR EARLY SELECTION OF HARDWARE--ETC(U)
DEC 80 B D HODGINS

UNCLASSIFIED NL

2 OF 2
AD A096302

END
DATE FILMED
4-81
DTIC

converted to floating point values. The Knuth Factor was applied only to the floating point operations instructions.

G. AN/UYK-20

Table IX.g

Times for the AN/UYK-20 were taken directly from the manufacturer's manual [19] except as indicated under comments.

The instruction set of the AN/UYK-20 does not provide for floating point operations. A method which approximates floating point operations was devised using the execution times of the appropriate fixed point operations. The floating point operations were determined as follows:

FL.P. ADD = 2 Fx.Pt. ADDS + 5 Shifts

FL.P. SUB = 2 Fx.Pt. SUBS + 5 Shifts

FL.P. MUL = 1 Fx.Pt. ADD of Exponents + 1 Fx.Pt. MUL of Mantissas + 10 Shifts for Normalization

FL.P. DIV = 1 Fx.Pt. SUB of Exponents + 1 Fx.Pt. DIV of Mantissas + 10 Shifts for Normalization

A penalty time was assigned to the fixed point (DP) MULT. The time calculated for the floating point MUL was used.

The Knuth Factor was not used on any of the instruction execution times calculated.

H. AN/UYK-7

Table IX.h

Execution times determined were taken directly from the manufacturer's manual [20]. All times shown assume 1.5 sec memory with operands not in same bank of memory as the instruction. The floating point (SP) MULT instruction execution

time was used for the fixed point (DP) MULT instruction execution time.

The Knuth Factor was not applied to any of the instruction execution times.

I. AN/AYK-14(V)

Table IX.i

Reference [21] was used to determine instruction execution times.

The AN/AYK-14(V) utilizes an Extended Arithmetic Unit (EAU) to enhance the execution speed of the floating point instruction for ADD, SUB, and MULT. The Knuth Factor was applied to these three instruction execution times.

J. Z-8000

Table IX.j

All information regarding timing was determined using ref. [22]. Instruction execution times for floating point instructions not included in the Z-8000's instruction set were determined by use of the method set forth for the AN/UYK-20 on page 96.

Fixed point (DP) execution times were not considered for the micros, because single precision operations are 16-bits in length which is the maximum length of all micros being considered. There are micros being developed now with 32-bit word lengths, and double precision operations. Evaluation of one of these machines will require that the double

precision execution times be included. The Knuth Factor was not considered for any of the instruction execution times for the Z-8000.

K. INTEL-8086

Table IX.k

Information regarding instruction execution times was provided by ref. [13]. The time required for an instruction to execute is the time required from beginning execution of an instruction that is in the instruction queue to the beginning of the next instruction execution.

Instruction execution is an asynchronous operation involving the Execution Unit (EU) and the BUS Interface Unit (BIU). The EU obtains each instruction to be executed from the Instruction object code queue (IOCQ) in the BIU. In determining the 8086 execution times it was assumed that the IOCQ was always full, and the EU never goes into a wait state.

The floating point instruction execution times were determined by method set forth for AN/UYK-20 on page 96. Fixed Point (DP) execution times were not considered. The Knuth Factor was not used for any instruction execution times of the INTEL-8086.

L. INTEL-8080

Table IX.l

Reference [13] was used to determine the instruction execution times of the 8080. Reference [13] provided the timings

for the basic instruction set while reference [12] provided algorithms from which approximate timing information was determined for the fixed point multiply and divide instructions in which the 8080 lacks in its instruction set.

Floating Point (SP) instruction timings were determined by method set forth for the AN/UYK-20 on page 96. Fixed Point (DP) timings were not considered. The Knuth Factor was not used for any instruction execution times of the INTEL-8080.

M. TI-9900

Table IX.m

Reference [13] provided instruction set timing information. All times indicated are maximum execution times.

Floating Point (SP) times were determined from method on page 96 for AN/UYK-20. Fixed Point (DP) times were not considered. The times indicated for the AND, and OR instructions are for immediate operations as that is all the instruction set allows. The Knuth Factor was not used for any of the instructions.

N. MC-68000

Table IX.n

Reference [23] was used to obtain all instruction timing information. All times listed include applicable operand fetches and stores. The Fixed Point (DP) instructions were not considered. Floating Point (SP) instruction timings were determined from method on page 96 for AN/UYK-20.

The Knuth Factor was not used for any of the instruction execution timings.

O. LSI 11/23

Table IX.o

Reference [24] was used to obtain all instruction timing information. The Fixed Point (DP) instructions were not considered.

The LSI 11/23 instruction set provides for floating point instructions. The times were determined by assuming the worst case, and taking into consideration all applicable notes which increased execution times. Mode 0 was assumed for all floating point instructions.

The general formula for determining execution times for the 11/23 instruction set is:

INST. TIME = BASIC TIME + SOURCE TIME + DESTINATION TIME

where,

Source Time (Double Operand)

Mode	Cycle	Time
0	0	0
1	1	1.12
2	1	1.12
3	2	2.25
4	1	1.42
5	2	2.55
6	2	2.55
7	3	3.67
Avg.	1.5	1.84

Destination Time	Cycles	Time
1. MOV, CLR, SCT, MFPS, MTPI (D)	1.50	2.27
2. CMP, BIT, TST	1.50	1.91
3. MTPS, MFPI (D), MUL, DIV, ASH, ASHC	1.50	0.99
4. BIC, BIS, ADD, SUB, SWAB, COM, INC, DEC, NEG, ADC, SBC, ROR, ROL, ASR, ASL, XOR	1.50	3.00

The Knuth Factor was not used for any instruction execution times.

TABLE IX

KEY TO MACHINE INSTRUCTION TIMES

FOR TABLES IX.a - IX.o

Symbol	Meaning
R or L	Right or Left
RX	Register-to-Memory
RR	Register-to-Register
Substitute	Time determined by using an alternate method when specified functional instruction not included in instruction set
Br.	Branch
SIP	Scientific Instruction Processor
SFT	Shift
EAU	Extended Arithmetic Unit
RW	Memory
See Attached	Refers to description of that machine in Appendix C
cc	Condition Code

TABLE IX.a

PDP 11/70

Functional Instructions	Instr. Used	Exec. Time (μsec)	Knuth Factor (μsec)	Comments
Fixed Point (SP)				
Add	ADD	2.94	----	
Subtract	SUB	2.94	----	
Multiply	MULT	4.10	----	
Divide	DIV	8.60	----	
Fixed Point (DP)				
Add	Routine	7.98	----	ADD, ADC, ADD
Subtract	Routine	7.98	----	SUB, SBC, ADD
Multiply	MULF	3.08	----	
Floating Pt. (SP)				
Add	ADDF	3.08	0.86	
Subtract	SUBF	3.08	0.86	
Multiply	MULF	3.08	0.86	
Divide	DIVF	3.08	0.86	
Logical				
Compare	CMP	2.19	----	
Shift	ASH	3.00	----	R or L 8 bit
And	BIT	2.19	----	
Or	BIS	2.94	----	
Control				
Load	MOV	2.31	----	Mem-To-Reg
Store	MOV	2.31	----	Reg-To-Mem
Br. Cond.	All	0.55	----	Avg.
Br. Uncond.	BR	0.72	----	
Inc. & Store Index	Routine	2.10	----	Inc. Write
Move	MOV (RR)	0.40	----	Reg. Mode
Index	ADD	3.06	----	Index Mode 6
I/O & Misc.				
I/O & Misc.	Avg. All	2.90	----	

TABLE IX.b

IBM 360/30

Functional Instructions	Instr. Used	Exec. Time (μsec)	Knuth Factor (μsec)	Comments
Fixed Point (SP)				
Add	AR	29.0	--	
Subtract	--	29.0	--	
Multiply	--	320.0	--	Substitute
Divide	--	600.0	--	Substitute
Fixed Point (DP)				
Add	--	65.0	--	Substitute
Subtract	--	65.0	--	Substitute
Multiply	--	320.0	--	Substitute
Floating Pt. (SP)				
Add	AER	65.0	--	
Subtract	--	65.0	--	
Multiply	ME	320.0	--	
Divide	DER	600.0	--	
Logical				
Compare	C	39.0	--	
Shift	SLL	58.0	--	Shift Left
And	NR	30.0	--	
Or	--	30.0	--	
Control				
Load	L	32.0	--	RX
Store	ST	32.0	--	RX
Br. Cond.	BC1, BC2	21.5	--	Avg.
Br. Uncond.	BAL	35.0	--	
Inc. & Store Index	AE	75.0	--	Substitute
Move	Load	22.0	--	RR
Index	--	75.0	--	Inc, Store
I/O & Misc.				
I/O & Misc.	Avg.	83.67	--	

TABLE IX.c

IBM 360/75

Functional Instructions	Instr. Used	Exec. Time (μsec)	Knuth Factor (μsec)	Comments
Fixed Point (SP)				
Add	AR	0.4	--	
Subtract	--	0.4	--	
Multiply	--	2.73	--	Substitute
Divide	--	6.87	--	Substitute
Fixed Point (DP)				
Add	--	0.85	--	Substitute
Subtract	--	0.85	--	Substitute
Multiply	--	2.10	--	Substitute
Floating Pt. (SP)				
Add	AER	0.85	--	
Subtract	--	0.85	--	
Multiply	ME	2.1	--	
Divide	DER	3.9	--	
Logical				
Compare	C	0.7	--	
Shift	SLL	0.6	--	Shift Left
And	NR	0.6	--	
Or	--	0.6	--	
Control				
Load	L	0.70	--	RX
Store	ST	0.82	--	RX
Br. Cond.	BC1, BC2	1.04	--	
Br. Uncond.	BAL	1.06	--	
Inc. & Store Index	AE	0.89	--	
Move	Load	0.40	--	RR
Index	--	0.89	--	Inc, Store
I/O & Misc.				
I/O & Misc.	Avg.	1.90	--	

TABLE IX.d

CDC 6600

Functional Instructions	Instr. Used	Exec. Time (μsec)	Knuth Factor (μsec)	Comments
Fixed Point (SP)				
Add	36	0.3	0.08	
Subtract	37	0.3	0.08	
Multiply	--	1.0	0.28	Substitute
Divide	--	2.9	0.81	Substitute
Fixed Point (DP)				
Add	--	0.4	0.11	Substitute
Subtract	--	0.4	0.11	Substitute
Multiply	--	1.0	0.28	Substitute
Floating Pt. (SP)				
Add	30	0.4	0.11	
Subtract	31	0.4	0.11	
Multiply	40	1.0	0.28	
Divide	44	2.9	0.81	
Logical				
Compare	Routine	1.9	0.53	13, 030
Shift	22, 23	0.3	0.08	R or L
And	11	0.3	0.08	
Or	12	0.3	0.08	
Control				
Load	50-57	1.1	0.32	
Store	50-57	1.2	0.32	
Br. Cond.	030-037	1.4	0.39	Br. in Stack
Br. Uncond.	04	1.4	0.39	
Inc. & Store Index	51	0.3	0.08	
Move	10	0.3	0.08	
Index	50	1.1	0.32	
I/O & Misc.				
I/O & Misc.	Avg.	0.54	0.15	

TABLE IX.e

CRAY-1

Functional Instructions	Instr. Used	Exec. Time (μsec)	Knuth Factor (μsec)	Comments
Fixed Point (SP)				
Add	060	0.05	0.014	
Subtract	061	0.05	0.014	
Multiply	032	0.0875	0.025	
Divide	---	0.0875	0.025	Substitute
Fixed Point (DP)				
Add	---	0.0875	0.025	Substitute
Subtract	---	0.0875	0.025	Substitute
Multiply	---	0.100	0.028	Substitute
Floating Pt. (SP)				
Add	062	0.0875	0.025	
Subtract	063	0.0875	0.025	
Multiply	064	0.100	0.028	
Divide	Routine	0.4875	0.137	070, 067, 064
Logical				
Compare	Routine	0.3375	0.095	046, 014
Shift	054, 055	0.0375	0.011	
And	044	0.025	0.007	
Or	051	0.025	0.007	
Control				
Load	12	0.35	0.079	Not in Buffer
Store	13	0.2125	0.060	Not in Buffer
Br. Cond.	014-017	0.3125	0.088	Worst Case
Br. Uncond.	06	0.3125	0.088	Worst Case
Inc. & Store Index	030, 11	0.250	0.070	
Move	024, 025	0.025	0.007	Worst Case
Index	176	0.225	0.063	
I/O & Misc.				
I/O & Misc.	Avg.	0.1029	0.029	

TABLE IX.f

HONEYWELL LEVEL 6/43

Functional Instructions	Instr. Used	Exec. Time (μsec)	Knuth Factor (μsec)	Comments
Fixed Point (SP)				
Add	ADD	1.47	----	
Subtract	SUB	1.47	----	
Multiply	MUL	8.54	----	
Divide	DIV	12.49	----	
Fixed Point (DP)				
Add	AID	1.84	----	
Subtract	SID	1.84	----	
Multiply	MUL	7.97	----	
Floating Pt. (SP)				
Add	SAD	3.75	1.05	SIP
Subtract	---	3.75	1.05	SIP
Multiply	SML	8.15	2.28	SIP
Divide	SDV	7.17	2.01	SIP
Logical				
Compare	CMR	1.73	----	
Shift	SAL, SAR	2.68	----	R or L
And	AND	1.34	----	
Or	OR	1.34	----	
Control				
Load	LDR	1.34	----	
Store	STR	1.57	----	
Br. Cond.	All	1.46	----	
Br. Uncond.	B	1.55	----	
Inc. & Store Index	INC, STR	3.14	----	
Move	SWR	1.80	----	
Index	INC	1.57	----	
I/O & Misc.				
I/O & Misc.	Avg.	3.62	----	

TABLE IX.g

AN/UYK-20

Functional Instructions	Instr. Used	Exec. Time (μsec)	Knuth Factor (μsec)	Comments
Fixed Point (SP)				
Add	22	0.75	---	
Subtract	20	0.75	---	
Multiply	26	3.80	---	
Divide	27	6.80	---	
Fixed Point (DP)				
Add	23	1.5	---	
Subtract	21	1.7	---	
Multiply	--	6.15	---	Substitute
Floating Pt. (SP)				
Add	Routine	2.80	---	2 ADD + 5 SFT
Subtract	Routine	2.80	---	2 SUB + 5 SFT
Multiply	Routine	6.15	---	ADD + MUL + 10 SFT
Divide	Routine	9.15	---	SUB + DIV + 10 SFT
Logical				
Compare	24	2.25	---	
Shift	14, 11	0.98	---	Avg.
And	30	0.75	---	
Or	31	0.75	---	
Control				
Load	00	2.25	---	
Store	10	2.40	---	
Br. Cond.	44-47	2.25	---	
Br. Uncond.	43	3.20	---	
Inc. & Store Index	15	2.40	---	
Move	70	4.50	---	
Index	05	2.25	---	
I/O & Misc.				
I/O & Misc.	Avg.	2.27	---	

TABLE IX.h

AN/UYK-7

Functional Instructions	Instr. Used	Exec. Time (μsec)	Knuth Factor (μsec)	Comments
Fixed Point (SP)				
Add	HA	1.00	---	
Subtract	HAN	1.00	---	
Multiply	HM	7.75	---	
Divide	HD	15.00	---	
Fixed Point (DP)				
Add	DA	3.00	---	
Subtract	DAN	3.00	---	
Multiply	--	10.00	---	Substitute
Floating Pt. (SP)				
Add	FA	6.25	---	
Subtract	FAN	6.25	---	
Multiply	FM	10.00	---	
Divide	FD	17.00	---	
Logical				
Compare	C	1.50	---	
Shift	HLC, HRZ	1.75	---	L and R
And	HAND	1.00	---	
Or	OR	1.50	---	
Control				
Load	LA	1.50	---	
Store	SA	1.50	---	
Br. Cond.	A11	1.50	---	
Br. Uncond.	JL	1.50	---	
Inc. & Store Index	SSUM	2.00	---	
Move	HLB	1.75	---	
Index	LXB	1.50	---	
I/O & Misc.				
I/O & Misc.	Avg.	2.68	---	

TABLE IX.i

AN/AYK-14(V)

Functional Instructions	Instr. Used	Exec. Time (μsec)	Knuth Factor (μsec)	Comments
Fixed Point (SP)				
Add	22	0.90	----	
Subtract	20	0.90	----	
Multiply	26	4.20	----	
Divide	27	8.30	----	
Fixed Point (DP)				
Add	23	1.10	----	
Subtract	21	1.10	----	
Multiply	56	6.90	----	
Floating Pt. (SP)				
Add	51	4.00	----	EAU
Subtract	50	4.00	----	EAU
Multiply	52	5.00	----	EAU
Divide	53	56.10	----	Worst Case
Logical				
Compare	24	2.00	----	
Shift	11, 14	4.10	----	L and R
And	30	0.90	----	
Or	31	0.90	----	
Control				
Load	01	2.00	----	
Store	11	1.90	----	
Br. Cond.	40	1.90	----	
Br. Uncond.	40	1.90	----	JR
Inc. & Store Index	15	1.40	----	
Move	01	0.90	----	
Index	LX	2.10	----	
I/O & Misc.				
I/O & Misc.	Avg.	5.04	----	

TABLE IX.j

Z-8000

Functional Instructions	Instr. Used	Exec. Time (μsec)	Knuth Factor (μsec)	Comments
Fixed Point (SP)				
Add	ADD	1.00	---	
Subtract	SUB	1.00	---	
Multiply	MULT	17.50	---	
Divide	DIV	23.75	---	
Fixed Point (DP)				
Add	--	0.00	---	Not Used
Subtract	--	0.00	---	Not Used
Multiply	--	0.00	---	Not Used
Floating Pt. (SP)				
Add	Routine	9.50	---	See Attached
Subtract	Routine	9.50	---	See Attached
Multiply	Routine	29.75	---	See Attached
Divide	Routine	36.00	---	See Attached
Logical				
Compare	CP	1.00	---	
Shift	SDA	8.25	---	
And	AND	1.00	---	
Or	OR	1.00	---	
Control				
Load	LDA	3.00	---	
Store	LD	2.75	---	
Br. Cond.	JP	2.13	---	If cc True
Br. Uncond.	JP	2.50	---	cc True
Inc. & Store Index	LDI	5.00	---	
Move	LD	0.75	---	
Index	Inc	3.50	---	
I/O & Misc.				
I/O & Misc.	Avg.	6.40	---	

TABLE IX.k

INTEL-8086

Functional Instructions	Instr. Used	Exec. Time (μsec)	Knuth Factor (μsec)	Comments
Fixed Point (SP)				
Add	ADD	0.60	---	
Subtract	SUB	0.60	---	
Multiply	MUL	24.80	---	
Divide	DIV	18.00	---	
Fixed Point (DP)				
Add	--	0.00	---	Not Used
Subtract	--	0.00	---	Not Used
Multiply	--	0.00	---	Not Used
Floating Pt. (SP)				
Add	Routine	18.80	---	See Attached
Subtract	Routine	18.80	---	See Attached
Multiply	Routine	42.20	---	See Attached
Divide	Routine	35.40	---	See Attached
Logical				
Compare	CMP	0.60	---	
Shift	SAL, SAR	1.734	---	
And	AND	0.60	---	
Or	OR	0.60	---	
Control				
Load	MOV	1.60	---	RW, DADDR
Store	MOV	1.80	---	DADDR, RW
Br. Cond.	All	2.00	---	Avg. True/False
Br. Uncond.	JMP	3.00	---	
Inc. & Store Index	INC, MOV	2.20	---	
Move	MOV	0.40	---	RWD, RWS
Index	INC	3.00	---	
I/O & Misc.				
I/O & Misc.	Avg.	2.682	---	

TABLE IX.1

INTEL-8080

Functional Instructions	Instr. Used	Exec. Time (μsec)	Knuth Factor (μsec)	Comments
Fixed Point (SP)				
Add	ADD	1.876	---	
Subtract	SUB	1.876	---	
Multiply	Routine	255.60	---	Attached
Divide	Routine	432.89	---	Attached
Fixed Point (DP)				
Add	--	0.00	---	Not Used
Subtract	--	0.00	---	Not Used
Multiply	--	0.00	---	Not Used
Floating Pt. (SP)				
Add	Routine	11.26	---	Attached
Subtract	Routine	11.26	---	Attached
Multiply	Routine	276.24	---	Attached
Divide	Routine	453.53	---	Attached
Logical				
Compare	CMP	1.876	---	
Shift	RLC, RRC	11.256	---	
And	ANA	1.876	---	
Or	ORA	1.876	---	
Control				
Load	LDA	6.097	---	
Store	STA	6.097	---	
Br. Cond.	All	4.69	---	
Br. Uncond.	JMP	4.69	---	
Inc. & Store Index	INC, MOV	5.628	---	
Move	MOV	2.345	---	
Index	INX	2.345	---	
I/O & Misc.				
I/O & Misc.	Avg.	3.8979	---	

TABLE IX.m

TI-9900

Functional Instructions	Instr. Used	Exec. Time (μsec)	Knuth Factor (μsec)	Comments
Fixed Point (SP)				
Add	A	9.99	---	
Subtract	S	9.99	---	
Multiply	MPY	19.98	---	Worst Case
Divide	DIV	5.99	---	Worst Case
Fixed Point (DP)				
Add	---	0.00	---	Not Used
Subtract	---	0.00	---	Not Used
Multiply	---	0.00	---	Not Used
Floating Pt. (SP)				
Add	Routine	37.30	---	Attached
Subtract	Routine	37.30	---	Attached
Multiply	Routine	47.29	---	Attached
Divide	Routine	33.30	---	Attached
Logical				
Compare	C	9.99	---	
Shift	SLA, SRA	17.316	---	L and R
And	ANDI	4.662	---	Immediate
Or	ORI	4.662	---	Immediate
Control				
Load	---	0.667	---	1 Machine Cycle
Store	---	0.667	---	1 Machine Cycle
Br. Cond.	All	3.33	---	
Br. Uncond.	B	5.328	---	
Inc. & Store Index	INC	51.282	---	
Move	MOV	9.99	---	
Index	INC	41.292	---	
I/O & Misc.				
I/O & Misc.	Avg.	7.356	---	

TABLE IX.n

MC-68000

Functional Instructions	Instr. Used	Exec. Time (μsec)	Knuth Factor (μsec)	Comments
Fixed Point (SP)				
Add	ADD	0.50	---	
Subtract	SUB	0.50	---	
Multiply	MULS	8.75	---	Worst Case
Divide	DIVS	19.75	---	Worst Case
Fixed Point (DP)				
Add	---	0.00	---	Not Used
Subtract	---	0.00	---	Not Used
Multiply	---	0.00	---	Not Used
Floating Pt. (SP)				
Add	Routine	3.00	---	Attached
Subtract	Routine	3.00	---	Attached
Multiply	Routine	12.50	---	Attached
Divide	Routine	23.50	---	Attached
Logical				
Compare	CMP	0.50	---	
Shift	ASR, ASL	2.25	---	
And	AND	0.50	---	
Or	OR	0.50	---	
Control				
Load	LEA	0.50	---	
Store	MOVEM	1.625	---	
Br. Cond.	Bcc	1.375	---	Avg. True/False
Br. Uncond.	BRA	1.250	---	
Inc. & Store Index	ADDI, MOVEM	3.625	---	
Move	MOVE	0.50	---	
Index	ADDI	2.00	---	
I/O & Misc.				
I/O & Misc.	Avg.	2.43	---	

TABLE IX.o

LSI 11/23

Functional Instructions	Instr. Used	Exec. Time (μsec)	Knuth Factor (μsec)	Comments
Fixed Point (SP)				
Add	ADD	6.56	---	
Subtract	SUB	6.56	---	
Multiply	MUL	27.35	---	
Divide	DIV	53.45	---	
Fixed Point (DP)				
Add	---	0.00	---	Not Used
Subtract	---	0.00	---	Not Used
Multiply	---	0.00	---	Not Used
Floating Pt. (SP)				
Add	ADDF	59.10	---	
Subtract	SUBF	59.10	---	
Multiply	MULF	102.75	---	
Divide	DIVF	104.25	---	
Logical				
Compare	CMP	5.47	---	
Shift	ASH	33.63	---	
And	BIT	5.47	---	
Or	BIS	6.56	---	
Control				
Load	MOV	5.83	---	RX
Store	MOV	5.83	---	RX
Br. Cond.	All	1.72	---	
Br. Uncond.	BR	1.72	---	
Inc. & Store Index	INC, MOV	10.55	---	
Move	MOV	5.83	---	RR
Index	INC	4.72	---	
I/O & Misc.				
I/O & Misc.	Avg.	6.33		

APPENDIX D

INSTRUCTION OVERLAPPING AND THE KNUTH FACTOR

One of the attractive features in the use of the IMSET as an evaluation tool is that a machine's ability to enhance its instruction executions through overlapping is taken into account. The IMSET is able to do this through use of a scaling factor derived from an article by Donald E. Knuth, ref. [11].

A. OVERLAPPING

In the most basic sense, overlapping is the ability of a computer to execute two or more instructions simultaneously thus executing more instructions within a given period of time. For example, the CRAY 1 utilizes twelve functional units for instruction executions. The CPU can continue issuing instructions for execution until it reaches a point where a required functional unit is not able to accept the instruction because it is already in execution. It is possible to have multiple executions taking place at the same time. Similar overlapping abilities exist in the CDC 6600 with its ten functional units. Special overlapping situations exist within machines such as the PDP 11/70, and the AN/AYK-14(V) which utilize separate hardware for only particular instructions. In these cases only a few instructions are able to be overlapped. For the PDP 11/70 and

AN/AYK-14(V) those instructions are the floating point operations. When a floating point instruction is encountered it is routed to a separate hardware unit for execution while leaving the CPU's arithmetic units free to continue execution of additional instructions. The instruction mix as a technique for evaluating computer thruput was not able to account for these overlap features in many of the later designed architectures, and thus it produced biased results.

B. KNUTH FACTOR

Knuth was interested in design of compilers which would produce optimal code for the most efficient program execution. He presented five levels of compilation ranging from level 0 to level 4. Level 0 compilations was straight code generation as would be produced by a classical one-pass compiler. Level 4 was considered to be the "best conceivable" code that could ever be imagined. Levels 1 through 3 fall at increasing levels of sophistication between levels 0 and 4. By analyzing Fortran programs that had been written, and looking at the sections of the programs which required the longest execution times Knuth attempted to pinpoint the areas where compiler optimization efforts should be directed to produce optimal compilation code, and maximum program execution speed. Results were then presented as a ratio of execution speeds with the five different levels of compiler optimization [11, pg. 32].

The Knuth Factor used to scale down the instruction execution times for overlap operations was determined by taking

the execution speed ratios for levels 0 to 3 as determined by Knuth's analysis. The ratio between level 0 and level 3 compilation was chosen for the following reason. A level 0 compilation is non-optimized compilation with no foresight as to optimization of instruction executions. Level 0 compilation would not separate consecutive instructions requiring the same functional unit for execution and parallelism would not be significantly exploited. Level 3 is a compilation level which produces machine-independent and machine-dependent optimizations. It is a level of sophistication which present day compilers are capable of obtaining. A level 3 compilation produces an optimization that attempts to maximize the use of available functional units. Consecutive instructions requiring the same functional unit would be separated so that the CPU could continue issuing instructions to available functional units without having to wait for a unit to become available.

The average speed ratio between level 0 and level 3 compilation was 3.62. Taking the reciprocal of this average produces 0.28 which is the scaling factor referred to in this thesis as the Knuth Factor.

The floating point ADD instruction execution time of the CDC 6600 is 0.4 microsecs. Multiplying (scaling) by the Knuth Factor (0.28) yields 0.11 microsecs as the time required to execute.

LIST OF REFERENCES

1. Lucas, Jr. H.C., "Performance Evaluation and Monitoring," Computing Surveys, V. 3, No. 3, P. 79-90.

2. Svobodova, L., Computer Performance Measurement and Evaluation Methods: Analysis and Applications, p. 52-56, American Elsevier, 1976.

3. IBM Technical Report TROO.2043, The Gibson Mix, by J.C. Gibson, p. 2, 18, June 1970.

4. Corsiglia, J., "Matching Computers to the Job - First Step Towards Selection," Data Processing Magazine, p. 23-27, December 1970.

5. Arbuckle, R.A., "Computer Analysis and Thruput Analysis," Computer and Automation, V. 15, No. 1, p. 12-19, January 1966.

6. Flynn, M.J., "Trends and Problems In Computer Organization," Information Processing 74, p. 3-10, 1974.

7. Knight, K.E., "Changes in Computer Performance," Datamation, V. 12, No. 9, p. 40-54, September 1966.

8. Raichelson, E. and Collins, G., "A Method for Comparing the Internal Operating Speeds of Computers," CACM, V. 7, No. 5, p. 309-310, May 1964.

9. Weitzman, C., Distributed Micro/Minicomputer Systems, p. 18-19, Prentice-Hall, Inc., 1980.

10. Honeywell Inc., St. Petersburgh Fla. Aerospace Div., ESD-TR-76-295, Secure Communications Processor Selection Trade Study, by C.H. Bonneau, March 1976.

11. Computer Science Department Stanford University Report CS-186, An Empirical Study of Fortran Program, by D.E. Knuth, p. 32, 1970.

12. Eckhouse, Jr., R.H. and Morris, L.R., Minicomputer Systems, 2nd ed., p. 138-139, p. 142-143, Prentice-Hall, 1979.

13. Osborne, A. and Kane, J., An Introduction to Microcomputers Volume 2, Adam Osborne & Associates, Inc., p. 4-32, p. 18-43, p. 20-68 through p. 20-73, September 1978.

LIST OF REFERENCES

1. Lucas, Jr. H.C., "Performance Evaluation and Monitoring," Computing Surveys, V. 3, No. 3, P. 79-90.

2. Svobodova, L., Computer Performance Measurement and Evaluation Methods: Analysis and Applications, p. 52-56, American Elsevier, 1976.

3. IBM Technical Report TROO.2043, The Gibson Mix, by J.C. Gibson, p. 2, 18 June 1970.

4. Corsiglia, J., "Matching Computers to the Job - First Step Towards Selection," Data Processing Magazine, p. 23-27, December 1970.

5. Arbuckle, R.A., "Computer Analysis and Thruput Analysis," Computer and Automation, V. 15, No. 1, p. 12-19, January 1966.

6. Flynn, M.J., "Trends and Problems In Computer Organization," Information Processing 74, p. 3-10, 1974.

7. Knight, K.E., "Changes in Computer Performance," Datamation, V. 12, No. 9, p. 40-54, September 1966.

8. Raichelson, E. and Collins, G., "A Method for Comparing the Internal Operating Speeds of Computers," CACM, V. 7, No. 5, p. 309-310, May 1964.

9. Weitzman, C., Distributed Micro/Minicomputer Systems, p. 18-19, Prentice-Hall, Inc., 1980.

10. Honeywell Inc., St. Petersburgh Fla. Aerospace Div., ESD-TR-76-295, Secure Communications Processor Selection Trade Study, by C.H. Bonneau, March 1976.

11. Computer Science Department Stanford University Report CS-186, An Empirical Study of Fortran Program, by D.E. Knuth, p. 32, 1970.

12. Eckhouse, Jr., R.H. and Morris, L.R., Minicomputer Systems, 2nd ed., p. 138-139, p. 142-143, Prentice-Hall, 1979.

13. Osborne, A. and Kane, J., An Introduction to Microcomputers Volume 2, Adam Osborne & Associates, Inc., p. 4-32, p. 18-43, p. 20-68 through p. 20-73, September 1978.

14. Digital Equipment Corporation, PDP 11/70 Processor Handbook, 1977-78 Edition, p. C-1 - C-14, 1976.

15. Solomon, Jr., M.B., "Economies of Scale and The IBM System/360," CACM, V. 9, No. 6, p. 436, June 1966.

16. Control Data Corporation, 6000 Series Computer Systems, Rev. 14 March 1974, 1974.

17. Cray Research, Inc., Cray 1 Hardware, 1978.

18. Honeywell, Inc., Honeywell Level 6 Minicomputer Handbook, Rev. 2, p. A-1 - A-8, September 1977.

19. Sperry Rand, AN/UYK-20 Military Computer, Undated.

20. Sperry Rand, AN/UYK-7 Military Computer, p. 44-73, Undated.

21. Control Data Corporation, AN/AYK-14(V), p. 32-36, Undated.

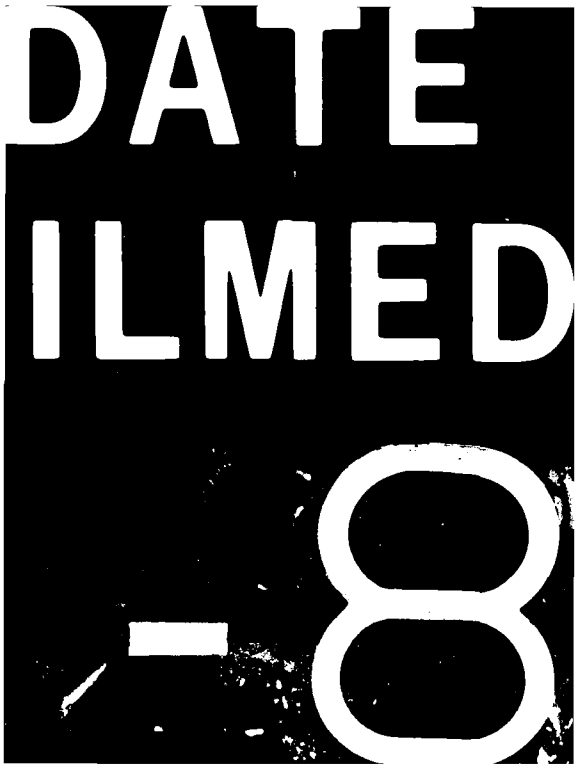
22. Zilog Incorporated, Z-8000 Development Module Hardware Ref. Manual, 1979.

23. Motorola, MC 68000 16-Bit Microprocessor, 2nd Ed., p. D-1 - D-8, January 1980.

24. Digital Equipment Corporation, Microcomputer Processor Handbook, p. 553-558, 1979.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
3.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
4.	Professor Lyle Cox, Jr., Code 52 Cl Department of Computer Science Naval Postgraduate School Monterey, California 93940	6
5.	Professor Uno Kodres, Code 52 Ko Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
6.	LCDR William D. Clayman, Code 80A Naval Undersea Warfare Engineering Station Keyport, Washington 98395	1


DATE
ILMED
-8