digital

interoffice memorandum

To: PMG distribution

CC: Performance committee

Date: 07 Dec 82
From: Mike Uhler
Dept: L.S.E.G.
DTN: (8-)231-6448
Loc/Mail stop: MRO1-2/E85
Net mail: UHLER at IO

Subject: Jupiter performance modeling

1.0 Introduction

Over the last few months, it has become apparent that the performance of the Jupiter CPU is less than had been originally estimated. Previous memos have discussed the limiting factors for better CPU performance, and it is understood, to a first approximation, what must be done to improve the performance of the machine. This memo proposes a strategy for improving performance, estimates the results of implementing the strategy, and lists costs and resources necessary to carry it out.

CPU only

2.0 Why performance modeling?

At present, the priorities for the Jupiter project indicate that we must ship a machine with a minimum performance of 2.3 times a KL10. In addition, if the FRS machine does have a performance near the minimum requirement, an enhanced machine must be shipped shortly thereafter. Enhancements that are currently planned are the APA and the so-called model B machine.

In order to be successful in any of these endeavors, several things must be understood, as follows:

- What is the current performance of the machine?
- How accurate are the models that are used to predict performance?

What is the impact of I/O on performance (vs. a KL)?
What is "performance" - interactive response, throughput, etc.? Anker has defined several categories
How will performance be affected as programs get larger?

How will changes in the d/s interface affect performance (vs a KL)?

- What constitutes a representative set of programs against which performance can be measured?
- What limits the performance of the machine?
- What can be done to increase the performance of the machine?
- What is the result on system performance if changes are made to the current design?
- What should be implemented in the APA and how much does this improve the performance?
- What should be implemented in the model B machine and how much does this improve the performance?

All of these questions must be answered in order to meet the goals of the Jupiter project. To do this, a performance modeling project must be undertaken that will address each of these items. This kind of modeling will not only answer the questions above, but will also reduce the risk to the Jupiter project by minimizing the surprises later on.

the KL? all DEC machines?

3.0 Performance predictions for various job mixes

In the past, the performance of a system has typically been measured with three kinds of loads, which can be loosely described as Fortran, Cobol, and general timesharing. The Fortran mix is characteristic of scientific applications and includes floating point and integer arithmetic operations. The Cobol mix represents the business ~~and~~ commercial applications and includes byte, string, and datatype conversion operations. The general timesharing mix (also known as logic mix) is more nebulous but includes such applications as text editors, compilers, debuggers, etc.

Because our machines are used in all three types of applications, it is important to be able to predict the performance of the Jupiter system on all three kinds of job mixes. In order to do this effectively, the modeling process used to predict the performance of the machine must include all three types of job mixes and not be limited to one. This slightly increases the complexity of the task, but it is both necessary and worthwhile.

The increasing use of extended addressing in both high-level languages and assembly language programs poses additional problems in trying to predict the performance of the machine. Experience on the KL10 has shown that pathological programs can show a severe performance degradation when moved from unextended to extended implementations. Therefore, performance predictions should not ignore the effects of extended addressing.

4.0 Minimum machine performance

The ultimate reason for doing performance modeling for Jupiter is to attain the system performance goal of 2.3 times a KL10. This goal deserves some discussion so that it is clear what it really means.

As indicated in section 3.0 above, there are three types of job mixes that will be modeled. What, then, does the phrase "2.3 times a KL10" really mean? The minimum performance number applies to the general timesharing (or logic) mix as measured on a Jupiter system. Fortran (without the APA) and Cobol job mixes will almost certainly run slower than the general timesharing mix. That is not to say that the Fortran and Cobol cases will be ignored in favor of improving only the general timesharing case. Rather, our goal is to improve all cases as much as we can within the other constraints of the project. However, if tradeoffs must be made, the general timesharing mix will have highest priority.

Note that the performance of the machine is a direct function of the machine cycle time. Changes in machine cycle time cause linear changes in the performance of the machine. Therefore, the determination of the cycle time is as important in predicting the machine performance as performance modeling.

Because there is inherent error in any performance modeling methodology (performance models are usually optimistic), the minimum performance prediction goal must be higher than the goal for the performance of the system. That is, the performance predictions must actually be higher than 2.3 times a KL10 to insure that the real machine will run at that performance. Therefore, the performance goal used for modeling the machine will be 2.5 times a KL10.

5.0 Performance improvements through performance modeling

The specific tasks of this proposal can be broken down into two categories. The first category makes the assumption that the performance data that exists is substantially correct. Using this data, simple microcode and hardware changes can be made to improve the performance of the machine. Section 6.1 below describes these changes in detail (also see the bibliography). These changes appear to have low risk to the current design and may produce a significant performance increase. Whether these changes alone will improve the performance enough to meet the goals is unknown. More extensive changes at this point would be risky because the accuracy of the data on which additional changes would be based is unknown.

The second category seeks to provide the additional data and the analysis tools necessary to understand what the performance of the machine really is. The data and tools produced will be used not only to direct additional changes to the design, if necessary, but

also to accurately predict the performance of the machine before it is built. This kind of modeling, which was not done during the design of the original machine, is critical if the performance goals are to be met in a predictable manner.

The proposal includes three areas of study: analysis and reduction of existing data, benchmark selection, and additional data gathering and analysis. Analysis of the existing data may help direct the task of making the simple microcode and hardware changes. Benchmark selection and additional data gathering and analysis are tightly coupled and are necessary both to accurately understand the performance of the machine, and to direct additional changes if that becomes necessary.

6.0 Specific tasks for performance modeling

This section discusses the specific tasks that should be undertaken for the Jupiter performance modeling project. For each of the four tasks listed, there are discussions of the goals, justification, benefit, cost and strategy for completing the task. There is no implied priority in the order in which the tasks are listed. In fact, several of these tasks should be undertaken in parallel.

6.1 Initial performance improvements

Based on preliminary investigations, it is known that there are certain classes of instructions that seem to limit the performance of the CPU. There are also certain microcode and "simple" hardware changes that can be made to increase the performance of the CPU with small redesign cost.

6.1.1 Goals

Make microcode and "simple" hardware changes to increase the performance of the machine, measure, with simulation, the resulting performance improvement for each change, and attempt to predict the change to the overall CPU performance.

6.1.2 Justification

In looking at the possibilities for improving the performance of the CPU, this type of change results in the smallest amount of hardware redesign. In addition, more extensive changes would be risky at this time because there isn't enough accurate data with which we can make design decisions.

6.1.3 Benefit

The actual performance improvement that will result from these changes is unknown. However, preliminary investigation indicates that it might be as high as 30%. Since the cost of this task is low compared to a larger hardware-oriented change, the return on investment is high.

How will we verify this

6.1.4 Cost

The major cost of this task is the manpower necessary to do the design, simulate the changes, implement the design changes in microcode and hardware, and measure the results of these changes. Secondarily, machine resources to do simulations, microcode compilations, etc. will be required.

6.1.5 Strategy

The strategy for completing this task may be broken down into five categories, as follows:

1. Evaluation and design. Evaluate the instructions that seem to be limiting factors to the performance of the machine. Determine what can be done in microcode and by adding minimal hardware to speed up these instructions. Based on preliminary investigation, it appears that the evaluations should be done in the following order:

   - o EA-calc speed-up in the EBOX. Evaluate the possibility of adding hardware to the EBOX to increase the speed of EA-CALC done by the EBOX. Such a change will improve the performance of byte, string, and XCT instructions, and indirect addressing.

   - o Byte pointer decode speed-up. Evaluate changes to the hardware (probably the micro-machine next-address dispatches) to make byte pointer decode faster. Such a change will improve the performance of byte and string instructions.

   The following items investigate improvements to the EBOX and IBOX microcode algorithms to make the instructions faster. Some minimal hardware changes may also be required.

   - o Other byte instruction speed-ups.

   - o BLT/XBLT speed-up.

   - o PUSHJ speed-up.

- o XCT speed-up.
- o String speed-up.

Design changes resulting from the investigation of the above list fall into the following areas:

- o EBOX EA-calc hardware additions.
- o EBOX and IBOX micro-machine dispatch changes.
- o Other minimal EBOX and IBOX hardware changes
- o EBOX microcode algorithm changes.
- o IBOX microcode algorithm changes (mostly the addition of new ICMDs).

I don't get it. Are you saying you've already investigated these?

2. Microcode implementation. Implement the EBOX and IBOX microcode changes that resulted from the design process.

3. LISP simulation. Implement the hardware changes that resulted from the design process in the LISP simulator. Use the modified simulator to insure that the CPU continues to implement the PDP-10 architecture. Then use the simulator as a tool to measure the performance of the machine.

4. Performance predictions. When the new performance for each instruction has been measured by the LISP simulator, combine that data with the best-guess machine cycle time and the simulated workload data that we have to predict the performance improvement.

5. Iterate. If the predicted performance is not 2.5 times a KL10, go back to step 1. How does this factor in conflicts?

6.2 Reduction of current OPHIST data

Instruction histogram data has been obtained from several sites with the OPHIST program. Reduction of this data is required if decisions are to be made on the basis of the data.

6.2.1 Goals

Reduce the large amount of raw data that exists such that we know correlations within each site and across sites. Produce an ordered list of "problem" instructions.

6.2.2 Justification

All OPHIST analysis to date has been done by manually correlating the data. There is no real confirmation that the order of investigation that was given in section 6.1.5 is correct. There is also minimal data on what is most important for the APA.

6.2.3 Benefit

This task provides a confidence factor that the priorities are indeed correct, especially in determining the sensitivity of the data that exists. It also produces an ordered list of the instructions that are important for both the APA and non-APA cases.

6.2.4 Cost

The primary cost of this task is the programmer necessary to write the data reduction programs. In addition, machine time is required to write, debug, and run the programs.

6.2.5 Strategy

The strategy for this task may be broken down into the following components:

1. Decide what correlations we need. It seems obvious that we need to know the sensitivity of data for each site, and across multiple sites.

2. Produce a list of the "most important" instructions for both APA and non-APA cases from the KC-weighted histograms. At present, such a breakdown for individual samples exists, plus a high-level summary for all samples. Additional breakdowns with more detail are required to direct the design changes.

3. Change the data reduction programs so that it is easy to change the machine cycle time and the performance of each instruction.

4. Attempt to define a "measure of goodness" using the OPHIST data so that we can predict the relative performance impact on the system of a change to the cycle time or the performance of a single instruction. Ideally, the result of this item will be to produce a single number that characterizes the performance of the machine with any workload. Changes in system performance as the result of changes to the cycle time or instruction performance would then be directly proportional to the change in the number.

6.3 Benchmark selection

In order to accurately predict the performance of a CPU, benchmarks that are representative of actual workloads must be available. This includes Fortran, Cobol, and general timesharing benchmarks.

6.3.1 Goals

Produce a list of representative benchmarks which can be used to predict the performance of Fortran, Cobol, and general timesharing job mixes. Must they run w/o JSYS's, or be smaller than n pages? How will they be used?

6.3.2 Justification

At present, there is no way to characterize the performance of the three kinds of instruction mixes that we are worried about. Up to now, performance predictions have been based on hand evaluation of OPHIST data.

6.3.3 Benefit

This task produces a representative set of benchmarks which can be used to measure and predict the performance of the CPU. In addition, these benchmarks allow us to evaluate changes to the cycle time and instruction performance.

6.3.4 Cost

The primary cost in completing this task is the manpower necessary to do the selections. There is also additional time involved in getting others to agree that the selections are indeed representative.

Make them run extended

6.3.5 Strategy

This task provides benchmarks in three areas, as follows:

- Opcode histograms. Opcode histograms (via OPHIST or other program) that are representative of Fortran, Cobol, and logic mix programs are needed. We may be able to construct composite opcode histograms out of the work done to reduce existing OPHIST data. Not for FOR or COBOL. No one has limited OPHIST to jobs running those things

- Programs as input to instruction simulators. These programs will be used to measure the impact of conflicts, cache and translation buffer hits, etc. See section 6.4.5 for more detail.

- Programs as input to the LISP simulator. These programs will be used to measure performance, instruction sequence interactions, etc. Because of the simulation rates and the limitations of the LISP simulator, such programs must run for less than 1 CPU second on a KL10, and issue no monitor calls.

Work has already been done by a number of groups in this area. There are a collection of programs commonly referred to as the "dirty dozen" that are allegedly representative of the general timesharing mix. Single and double precision Whetstones exist that provide some indication of the Fortran performance. The monitor group has a collection of benchmarks that may be helpful. Some work was done on Cobol performance for the Dolphin project and a composite Cobol program was constructed that was supposed to be representative of what typical Cobol programs do.

If priorities must be assigned, it is most important to select representative benchmarks for the general timesharing mix. The performance goals are based on that mix and these benchmarks will be used to decide whether additional work is necessary to increase the performance of the machine.

6.4 Additional data gathering and investigation

Due to the scarcity of performance data, it is critical to the success of the project to gather and evaluate additional data. This process is important not only for the FRS machine, but also for the APA design and the design of the model B machine.

6.4.1 Goals

Produce additional data and analysis tools that will increase the accuracy of our performance predictions. Quantify the effects of extended addressing, indirect addressing, IBOX conflict, IBOX flush, IBOX prefetch efficiency, translation buffer conflicts, and cache hit.

6.4.2 Justification

At present, performance estimates are based on OPHIST results only. There is general agreement that the OPHIST results approximate the characteristics of real workloads, but only minimal attempts have been made to confirm this speculation. In addition, there is no actual data on the impact of things like

IBOX efficiency, conflicts, the translation buffer, etc. This part of the modeling process is the most critical in understanding the real performance of the system.

6.4.3 Benefit

By producing additional data and analysis tools, performance predictions will be more accurate. It will also quantify the (currently unknown) effects of the IBOX.

6.4.4 Cost

Of all the components of the performance modeling proposal, the costs associated with this section are the largest. Completion of the items listed below will require one or more people and significant machine resources. The simulations are not possible with the load averages on existing machines.

6.4.5 Strategy

The strategy for this task is broken down into multiple areas, as follows:

1. Additional OPHIST data. OPHIST data must be gathered from additional sites whose typical load is Fortran, Cobol, or general timesharing. More sites will increase the confidence in the accuracy of our performance predictions. This is especially true if there is a good correlation between sites with similar workload characteristics. An attempt should be made to select at least two sites whose typical workloads are general timesharing, Fortran, and Cobol. At least one week of data is required from each site to smooth out the day-to-day variations in load.

2. TRACKS microcode on the KL10. TRACKS microcode is required for several reasons, as follows:

   - Verification of OPHIST results. In one data gathering mode, TRACKS microcode will keep an opcode histogram of instruction execution. By running OPHIST on a machine with TRACKS opcode counting enabled, we should get an indication of the validity of the OPHIST measurement technique. This is particularly important since many decisions are being based on the OPHIST data.

   - Exec mode measurements. Parts of the monitor run with the PI system off. Since OPHIST uses the interval timer as a stimulus, it can't sample those areas of the monitor. The exact impact of this is unknown, but there is a general

feeling that certain important parts of the monitor (e.g., indirect references to the CST) are being masked.

- o PC traces. In another data gathering mode, TRACKS has the ability to generate PC traces. These traces could be used as input to a program for analysis of instruction sequences.

3. Conflict analysis. At present, there is no data on the effects of instruction conflicts. By using CONF20, a program written for the Dolphin project, with "typical" programs, data can be gathered concerning conflicts. At present, CONF20 will only run in section zero, and it needs some work. It should be modified to run in a non-zero section and analyze multi-section programs.

4. Enhancements to the LISP simulator. In order to determine the effects of IBOX flushes and the efficiency of the IBOX prefetch algorithms, the LISP simulator must be modified to keep more data about the programs that it is simulating. Counters can be installed that will keep track of IBOX flushes, conflicts, guess-wrong, and the average number of instructions ahead that the IBOX is fetching. Because of simulation ratios, the selection of programs may be difficult.

5. Translation buffer and cache hit analysis. By using a program similar to SIM20, address traces can be obtained for representative programs. These traces can then be analyzed by an existing cache simulator to give us an indication of the effectiveness of the translation buffer and the cache. If SIM20 is used to provide the address traces, it must be modified to run in a non-zero section and measure multi-section programs.

   This item is particularly important because we have no up-to-date data on the effects of extended addressing on the translation buffer and cache organizations. A thorough analysis of this topic also requires data on context switch time. First-order cache models assume that the cache has reached steady-state. If the context switch rate is too high, the cache doesn't have a chance to reach steady-state and the cache hit predictions will be too high.

I think you should request the services of an experienced model-builder, who can not only

7.0 Ordering the tasks

Under the assumption that the existing performance data is correct, we can start the design and implementation process for the simple hardware and microcode changes immediately. In parallel with this, it is important to do the data reduction on the existing OPHIST data so that we can measure the relative effect of the design changes. Although it may not be entirely

model all data, but make it easy to test possible changes to the design by simple changes in the model

accurate, the OPHIST data is all that we have with which to measure performance. In addition, OPHIST data can be used as an indication of relative performance changes when a design change is made.

Benchmark selection can be done somewhat asynchronously to the design changes and the reduction of the OPHIST data. It would be useful to have some representative benchmarks available when the simulator and microcode changes are completed so that some preliminary performance predications can be made. Benchmarks must be available in order to do any significant data gathering, beyond what already exists.

Most of the additional data gathering and analysis is independent of the simple design changes but is dependent on benchmark selection. Completion of this task is required before any accurate performance predictions can be made. In addition, no significant hardware changes can be made, beyond those outlined in the above sections, until this task is complete. Besides the required manpower, this task depends on machine resources that are not currently available.

8.0 Conclusions

No serious performance modeling was done during the design of the original machine. As a result, the performance is less than we expected and some redesign is being done. This memo proposes a performance modeling project that will aid us in making decisions about the design changes, predict the performance of the machine, and minimize the risk to the project.

The importance of this project should not be underestimated if we are to meet our performance goals in a predictable way. In addition, no significant hardware redesign should take place without understanding the effect on system performance of that change.

The data and tools that are produced by the performance modeling project will be used not only for redesign of the FRS machine, but also as direction for the design of the APA and the model B machine.

The ultimate machine performance is directly related to the cycle time. Determination (and minimization) of the cycle time is critical to an accurate performance prediction and hence, to machine performance. That determination must be given equal priority to performance modeling.

how?

APPENDIX A

BIBLIOGRAPHY

The following bibliography lists documents relating to PDP-10 performance. Copies of these documents are available from me. The list is in chronological order.

1. Newman, Michael, "Improved performance String Move", Digital interoffice memorandum, June 23, 1977. Analysis of improved string performance.

2. Murphy, Dan, "Page Refill Timing Data", Digital interoffice memorandum, December 6, 1977. Analysis of page refill operations on the KL10 under KL10 paging.

3. Hess, Ted, "Possible 2080 Performance", Digital interoffice memorandum, February 29, 1980. Simplistic predictions of the 2080 performance based on work done on the Dolphin project.

4. Hess, Ted, "Possible 2080 Floating-point Performance", Digital interoffice memorandum, March 5, 1980. Simplistic predictions of the 2080 floating point performance (with and without an APA).

5. Miller, Arnold, "2080 extended addressing performance", Digital interoffice memorandum, May 24, 1982. Thoughts on the impact of extended addressing on the performance of a Jupiter system. Concentrates on the effects of the translation buffer organization and the effects of indirect addressing.

6. Manley, Dwight, "Jupiter workload analysis", Report on performance analysis, September, 1980. Analysis of Fortran and Cobol programs to predict the efficiency of the IBOX.

7. Uhler, Mike, "Jupiter Performance", Digital interoffice memorandum, September 6, 1982. Describes the preliminary results of the performance analysis done for Jupiter including a simple performance model.

8. Nixon, David, "Performance prediction report for Jupiter", Digital interoffice memorandum, October 5, 1982. Summary and

analysis of the OPHIST data.

9. Nixon, David, "Performance Prediction Method for Jupiter", Digital interoffice memorandum, November 4, 1982. Discussion of the methodology used to gather and analyze the OPHIST data.

10. Uhler, Mike, "Minutes of the 11/24/82 Performance Committee Meeting", Digital interoffice memorandum, November 26, 1982. Describes the proposed structure of the Jupiter performance committee. Lists a hierarchy of proposed solutions and the resources necessary to support the investigations.

digital

interoffice memorandum

To: Peter Hurley
Bill McBride

CC: Judy Hall
Don Hooper
David Nixon
Pat Sullivan

Date: 06 Sep 82
From: Mike Uhler
Dept: L.S.E.G.
DTN: (8-)231-6448
Loc/Mail stop: MRO1-2/E85
Net mail: UHLER at IO

Subject: Jupiter performance

1.0 Introduction

Over the past few weeks, Judy Hall and David Nixon have been gathering workload data from several "typical" systems in an attempt to characterize the performance of the Jupiter. The data gathering techniques being used are fully described in a memo by David Nixon. In looking at the initial results, I have developed a simple model that provides a first-order approximation of the performance of the Jupiter. In addition, we have identified what we believe to be a significant performance bottleneck in the EBOX speed of certain instructions. This memo describes the model, identifies certain classes of instructions that appear to be performance bottlenecks, and makes recommendations about possible microcode/hardware solutions to these bottlenecks.

2.0 The performance model

In looking at the original workload data, I noticed that the set of executed instructions seemed to fit into three broad categories as follows:

1. Instructions whose KC/KL ratio is less than 3.

2. Instructions whose KC/KL ratio is between 3 and 6.

3. Instructions whose KC/KL ratio is greater than 6.

After some simple calculations on the preliminary data, I concluded that the percentage of executed instructions in each class was approximately:

Class	Percentage of instructions in class
1 (< 3 times KL)	25%
2 (3 - 6 times KL)	25%
3 (> 6 times KL)	50%
	100%

This means that half the executed instructions run at least 6 times faster on a Jupiter than the same instructions on a KL10. Intuition would lead one to believe that the performance of the Jupiter would be outstanding. But let's calculate the predicted performance of the machine using this model and choosing one "average" number for each class.

Class	KC/KL ratio	Percentage	Weighted time
1	1.5	25%	0.167
2	3.0	25%	0.083
3	8.0	50%	0.063
			0.313

The "weighted time" column was computed by multiplying the percentage for each class by the inverse of the KC/KL ratio, e.g.,

$$0.25*(1/1.5) = 0.167$$

This number gives the time, in KL units, that the class of instructions would take to execute on the Jupiter. The sum of the column gives the total time, again in KL units, that all instructions would take on a Jupiter. The inverse of this number gives the predicted performance of the Jupiter. In this case, the predicted performance ratio is 3.2.

If the model says that half the instructions run at 8 times a KL10, why is the overall predicted performance only 3.2? Let's look at the weighted time column for the answer. The class 3 instructions, which amount to 50% of the executed instructions account for only 20% of the execution time (0.063/0.313). The class 1 instructions, on the other hand, which amount to only 25% of the executed instructions, account for over 50% of the

$KL/KC = \sum \text{percentage} * \frac{KL}{KC} \text{ ratio}$

$KC/KL = \frac{1}{KL/KC}$

execution time. This non-intuitive behavior means that the instructions that are relatively slow on the Jupiter make up a large part of the total execution time even if they are a relatively small percentage of the total instructions executed.

Let's see what happens if we adjust the KC/KL ratio for one class. First, assume that the class 3 instructions actually run at 4 times a KL10 (as would be the case if the EBOX were constantly waiting for the IBOX to finish setting up an instruction).

Class	KC/KL ratio	Percentage	Weighted time
1	1.5	25%	0.167
2	3.0	25%	0.083
3	4.0	50%	0.125
			0.375

Predicted performance: 2.7

A 50% change in the performance of the class 3 instructions only makes a 16% change in the performance of the overall machine.

Let's see what happens if we change the speed of the class 1 instructions instead by assuming that they are only 1.2 times a KL10.

Class	KC/KL ratio	Percentage	Weighted time
1	1.2	25%	0.208
2	3.0	25%	0.083
3	8.0	50%	0.063
			0.354

Predicted performance: 2.8

A 20% change in the speed of the class 1 instructions made a 12% change in the performance of the overall system and the machine is now spending 59% of the EBOX compute time processing these instructions.

2.1 Significance and accuracy of the model

In the beginning, the development of the model was an attempt to predict the performance of the Jupiter using a very simply, easy to change model of the machine. From that beginning, it has developed into a tool for understanding why the performance of the machine isn't what we thought it should be. As the calculations

in the previous section demonstrate, one can change the predicted performance of the machine by similar amounts, either by making large changes in the performance of the fast instructions or by making small changes in the performance of the slow instructions. By using a simple model, it is much easier to understand this non-intuitive behavior.

Since I developed the original model, I have seen more workload data that makes me believe that the model is actually optimistic. I believe that the typical KC/KL ratios for classes 2 and 3 are reasonably accurate at 3 and 8 respectively. If this assumption is correct, the "average" KC/KL ratio for class 1 must be unrealistically large, even when it is set at 1.2. Further analysis is required to determine the correct numbers to be used in the model.

As with all simple models, this one doesn't exactly predict the true performance of the machine. It is, however, a first-order approximation of the performance characteristics of the Jupiter and it does demonstrate that the slow EBOX instructions will be the limiting factor in the speed of the overall machine.

3.0 Characterizing the slow instructions

In looking at the workload data from various sites that has been sorted by KC weight (i.e., the percentage of EBOX compute time for each instruction), one observes that the relative position of each instruction changes for each site. However, the same instructions always seem to appear somewhere near the top of the list. These instruction classes are listed below. The table is given in alphabetical order and does not reflect the actual order of frequency.

1. BLT

2. Byte (LDB, IDPB, etc.)

3. Floating point (both single and double)

4. PUSHJ/POPJ

5. String (MOVSLJ, CVTxxx, etc.)

6. XCT

The data that we have indicates that these six instruction classes account for 30 to 80 percent of the total EBOX compute time. As such, changes in performance of these instructions could have a significant impact on the overall performance of the machine. I believe that we should be concentrating on optimizing the

performance of these instructions.

4.0 Possible microcode/hardware optimizations

I have done a cursory investigation of each of these classes of instructions and I believe that certain changes are possible that could significantly increase the performance of certain classes. This section is broken down into subsections, one for each class of instruction. Each subsection describes the results of the investigation and gives recommendations for each class.

4.1 BLT

BLT appears to spend a significant amount of time loading the read/write address into EA buffer. Changing the EBOX and IBOX microcodes to use new functions which allow more overlap in the read/write of words appears to make a significant difference. There may also be some potential in using two-word reads to get source data.

Estimated performance improvement: 2.0-3.0.

4.2 Byte (LDB, IDPB, etc.)

A quick count of microcycles seems to indicate that byte instructions spend their time doing the following:

1. Byte pointer typing, validation - 40%

2. Byte pointer eacalc - 30%

3. Byte manipulation - 30%

The first two items have the most potential for improvement. Adding new dispatches may improve the ability to determine the byte pointer type quickly. Improvement in the eacalc time (see XCT below) could improve the byte pointer eacalc time. Additional hardware to decode the byte pointer and perform the byte manipulation would be required to make a drastic change in the performance of these instructions.

Estimated performance improvement: 1.1-2.0.

4.3 Floating point (both single and double)

Because of the lack of a 72 bit data path in the EBOX, there isn't much that can be done in microcode to improve these instructions. The addition of the FPA should improve the performance of these instructions significantly.

Estimated performance improvement (with FPA): 2.0-5.0

4.4 PUSHJ/POPJ

PUSHJ spends most of its time determining what to store in the stack word and how to update the stack pointer. Some improvement can be gained by adding new dispatches to allow the microcode to check more conditions in parallel. We may also gain some improvement from a change to the IBOX microcode.

Estimated performance improvement: 1.1-1.8.

The POPJ instruction has no extraneous microcycles as it is presently coded. I see no real improvement possible for this instruction without hardware changes.

4.5 String (MOVSLJ, CVTxxx, etc.)

I know the least about this class of instructions. From what I do know about them, it appears that they have significant potential for improvement. Special casing certain common operations, avoiding the eacalc on every byte (if possible), and careful hand optimization could make a large difference. We may also be able to take advantage of any changes that improve the performance of the byte instructions.

Estimated performance improvement: Unknown

4.6 XCT

Most of the time spent in the XCT instruction is spent performing the eacalc on the executed instruction and fetching its operands. Improving the speed of the eacalc subroutine could significantly increase performance of this instruction. Unfortunately, only additional hardware will make this possible. Improving the eacalc speed will also benefit byte and string instructions and IBOX traps to EBOX for indirect instructions. An IBOX that processes 1

level indirect doesn't solve the problem for XCT and byte instructions; the EBOX eacalc routine must be made faster.

Estimated performance improvement (with hardware): 1.5-2.Ø

4.7 Impact on system performance

We have not yet used the estimated performance numbers given above to analyze the impact of making each change on overall system performance. This work should be completed in the next week or two and that data will give us an ordered list of optimizations to make.

5.Ø Summary of findings

Given this data, one can make certain statements about the performance of the machine, both in general and in specific terms.

The performance of a machine is a function of ALL the instructions executed on that machine. Significantly increasing the performance of one class of instructions while ignoring another class tends to result in a machine whose performance is bound by the class that was ignored. Better overall system performance is achieved by increasing the performance of all instructions by approximately the same amount.

The primary performance bottleneck is the EBOX compute time of the slow instructions. Typically, EBOX processing of this class, which amounts to approximately 25% of the executed instructions, takes 6Ø to 8Ø percent of the system.

Microcode changes can be made to significantly increase the performance of the machine by optimizing certain of the critical instructions. More analysis must be done to predict the overall change to system performance.

Certain hardware changes can be made to further increase the performance of the machine. These changes should be made with careful attention given to the benefit/risk tradeoff. Adding new dispatch bits so that the microcode may check several conditions in parallel may prove to be the most beneficial change.

Instructions whose KC/KL ratio is 3 or greater are not worth optimizing at this point since the resulting change in performance is negligible.

The efficiency of the IBOX seems to have only second-order effects on the overall performance of the system. This has been confirmed

with initial IBOX simulation data. It is possible that this could change if the performance of the slow instructions is significantly improved, although the available data doesn't seem to indicate that this will happen.

6.0 Recommendations

The only realistic way to solve these problems is with a top-down approach. We cannot afford to implement solutions and then design them. We must evaluate all changes from a system viewpoint and know in advance what impact those changes are going to have on the performance of the system.

I suggest forming a working group consisting of knowledgeable people in the areas of architecture, performance, microcode, and hardware design whose charter would be to oversee any changes that are made.

There are indeed problems with the performance of the Jupiter CPU. Fortunately, there are also solutions to quite a few of these problems and the potential exists to significantly increase the performance of the machine.

+---------------+
! d i g i t a l ! I N T E R O F F I C E M E M O R A N D U M
+---------------+

TO: ASK
Bill McBride
Peter Hurley
Dave Braithwaite
Rich Fiorentino

DATE: May 24,1982

FROM: Arnold Miller

DEPT: LCG S. E. Dept.

LOC: MR1-2/L10 EXT: 231-6473

NET: KL2102:: or MRVAX::

SUBJ: 2080 extended addressing performance

1.0 Executive summary

Much has been said about the performance concerns of Jupiter. One of the most important elements of this concern is the performance of "extended addressing" on the Jupiter. This is so because all of our products either now use extended addressing, or plan to use it soon. Furthermore, our customer's expectations for an efficient and timely technology for handling large data bases is quite high.

Extended addressing performance concerns derive from two distinct problems: the caching of page pointers by the MBOX and the cost of indirect addressing.

We've already done a great deal of research into the page cache problem. The design of the 2080 MBOX and page refill algorithm reflect what we have learned. No doubt more could be done, but we don't understand the benefits and weaknesses of the new design sufficiently well to undertake more design work. The KL-10 pager, however, could benefit by some of this knowledge.

Indirect addressing performace on the 2080 is quite another story. Whereas indirect addressing on the KL-10 operates acceptably, the projected performance of indirect addressing on the 2080 raises significant concerns. The architecture of extended addressing relies heavily on indirection and both the monitor and user-level programs reflect this architecture.

2.0 Page cache

The KL-10 has a well-publicized deficiency in the caching of paging information. The KL-10 can cache up to 512 valid paging translations, but entries are replaced four-at-a-time (block

size of four). Therefore the effective caching is reduced somewhat by frequent conflicts. This limited cache size affords many opportunities for programs to thrash by having two of more of the addresses involved in executing an instruction conflict. An "address conflict" occurs when two addresses occupy the same location in the paging cache (also known as the translation buffer); or in other words caching one of the addresses invalidates the entry for one or more of the other addresses. Since caching is page-by-page, conflicts can easily remain in force over many instructions or many data words.

In an effort to ameliorate this, the 2080 has a much larger paging cache with a different organization (block size of one). Also, other measures that minimize the need to clear the cache have been provided. However, pathological conditions may still arise. In general, these pathological conditions are eliminated by using a sufficiently high degree of associativity in the cache. However, the 2080 page cache, like the KL-10 cache, is only one-way associative (A particularly interesting observation is that the data cache on both machines is four-way associative whereas the page cache is only one-way associative). In light of this, we have recommended that the 2080 EBOX microcode provide some additional caching of page pointers or section pointers to smooth out these pathological cases. The recent decision to extend the life of the KL-10 raises the question of whether we can do the same sort of caching on the KL-10.

Before any additional measures are considered, it seems prudent to commission a study of the problem to produce either analytical or empirical support for new features (An article in a recent issue of Computer Magazine reported on the performance characteristics of the VAX11/780 paging cache, including studies on various degrees of associativity. The conclusion seems to be that increasing the cache size has the same benefit as increasing the associativity, but the architectural differences between the VAX and the 2080 could be significant. In the absence of any other data, we can only assume that the data applies to the 2080 as well as to the VAX. The article did not address "pathological conditions").

3.0 Indirect addressing

The second problem, and one that we have yet to find an adequate compromise for, is the performance of indirect references.

The extended addressing architecture provides two techniques for inter-section or global addressing: simple indexing and indirection. Indexing allows one to address a contiguous region of 256K words of virtual memory, plus or minus 128K from the base indicated by the index register. In other words, indexing computes a full 30-bit address by adding the value in the index register to the sign-extended value found in the address field of the instruction.

Another interesting property of global indexing is that the index register must contain the base address of the data, and the instruction word must be used as the offset. This is contrary to the addressing and programming practiced up until now on PDP-10s. This new way of addressing is brought about because the instruction has only eighteen bits in which to express an address, but a register has thirty bits. Therefore, in order to address data that may be located anywhere in the extended address space, one must swap the roles of the instruction and the index register.

A problem created by this is that the familiar technique for writing loops becomes invalid. That is, the code sequence:

```
MOVE AC,BASE(ACX)

   ...

AOBJN ACX,LOOP
```

is no longer appropriate as ACX can no longer contain

```
-COUNT,,OFFSET
```

Instead, loops require the use of two or more registers if indexing is the addressing choice. For example:

```
HRRZ ACX1,ACX
ADD ACX1,BASE
MOVE AC,0(ACX1)

  ...

AOBJN ACX,LOOP
```

Other instruction sequences are possible, but this example is representative of the nature of the difference.

Indirection allows one to address data stylistically similarly to the traditional methods. That is, indexing is always a positive (or negative) offset from the unsigned base and a single index may be used for offset addressing and loop control.

Therefore one can write:

```
MOVE AC,@[EFIW BASE(ACX)]

   ...

AOBJN ACX,LOOP
```

3.1 Discussion

Despite the architectural bent in favor of indirection, and the additional clarity afforded by the technique, the 2080 was apparently not designed with this sort of addressing in mind. Indirection will effectively defeat much of the advantage of the IBOX prefetch and result in a significant increase in execution time for any instruction employing it. This is so because the IBOX is not equipped to decode indirect references, and the EBOX is not sufficiently facile at effective-address calculations to offset this deficiency.

Up until now, all of the extant software projects had assumed the use of indirect addressing. TOPS-20 has already been modified to reference its paging data base with indirection, and more and more instances of indirection appear in the monitor as time goes by.

The languages had planned to use indirection to reference large data bases. Again, this seemed the right choice because of the architectural direction and the ease of substituting indirection for the existing simple indexing methods.

3.2 Impacts

(All 2080 performance figures are based on "best guesses". The variation between these values and the actual machine performance may be significant- as much as 30%. See section 3.3 for more information).

As best we can understand, the use of indirection will be five times more expensive in execution time than the use of indexing. Or looked at it in comparison with indirection on the KL-10, simple instructions that use it will execute in about the same time as the same instruction, including indirection, on the KL-10 (see chart below). Also, the instructions required to implement indexing run no faster, and in some cases slower, on the KL-10 than the indirect addressing style on the KL.

This represents the directly measurable differences. However, increased code sizes, within loops or not, will affect cache hits, in the data cache, the paging cache and the IBOX. These effects are second order and hard to predict.

It may be possible to avoid using indirection in the languages, but the cost of larger loops, greater complexity in the compilers (more use of volatile registers) and the variance in characteristics between the KL-10 and the 2080 are significant unknowns. The current plans are to use indirection until and unless clear reasons demand a change in plans. The cost to switch to indexing in FORTRAN has been estimated at two man-months.

The monitor code has already been done and is working. Replacing the uses of indirection with indexing, while not out of the question, will take considerable time and effort. To

have to commit resources to a project that provides no direct improvement in the product at the expense of potentially marketable improvements, would be unfortunate. And, as with the languages, the differences between the KL-10 and the 2080 may well mean that the code is still less than optimal for one of the processors.

The time to convert the "performance critical" uses of indirection to indexing is comparable to that for FORTRAN: two months.

3.3 Instruction timing chart

This chart shows the time (indirection) to execute a single MOVE instruction using "global indirection" for addressing. An example of this is in section 3.0 above. It also shows the time (indexing) to execute a sequence of instructions that achieves the same result as indirect addressing, but with "global indexing" as the addressing mode. Again, section 3.0 gives an example of such a code sequence.

	2080	KL-10
indirection	1100 nsecs	900 nsecs
indexing	200 nsecs	1300 nsecs

The values for the 2080 were derived by "counting cycles", estimating the cost of IBOX conflicts and averaging the various instruction sequences that could be used. Therefore this is not the "best case" performance for the 2080. Furthermore, the numbers represent only an "educated guess" and could vary as much as 30% from the "true values".

The KL-10 values were derived from instruction timings on KL2102 using "the best case" instruction sequence given in section 3.0.

To put this in perspective, let's assume the 2080 is 5X the performance of a KL-10. Furthermore, let's assume that an instruction that uses indirect addressing will run at the same speed as the same instruction on the KL-10. Furthermore, any instruction that does not use indirection, will run at 5X the same instruction on a KL-10 (this is, admittedly, a questionable assumption). If 20% of the instructions in a program use indirection, then the effective speed of this program will be 2.8X a KL-10, or a 45% loss in throughput. If 10% of the instructions use indirection, then there will be a 27% loss. Presently, the uses of indirection are rare, but as applications take advantage of extended addressing, use of indirect addressing will grow. A first-order guess that 20% of the instructions in a typical FORTRAN application using array

data seems appropriate.

Cost of indirection
(ideal power factor is 5)

% indirect instructions	power factor	% loss to ideal
30	2.3	54
20	2.8	45
15	3.2	37
10	3.6	27
5	4.2	17

3.4 Other "indirection" problems

Certainly, the above-stated problem with writing loops is the most pronounced. However, all cases of addressing are affected by the performance of indirection.

For example, consider a subroutine call. Normally this would look like (all timings are subject to the same caveat as given in section 3.3)

```
CALL SUB                    (528 nsecs)
```

However, if the routine being called may be in another section from the call site, we would be inclined to write:

```
CALL @[EFIW SUB]            (1628 nsecs)
```

However, the cost of the indirect address calculation might well dictate code of the form:

```
XMOVEI SAC,SUB              (968 nsecs)
CALL 0(SAC)
```

or

```
MOVE AC,[SUB]               (682 nsecs)
CALL 0(SAC)
```

The latter form incurs a penalty of approximately 154 nsecs over the non-indexed and non-indirect form, the middle form a penalty of 440 nsecs and the from using indirection of 1100 nsecs. Therefore, the penalty for indirection may be as high as an order-of-magnitude greater than that for indexing.

The KL-10 will execute the indirect form of the call slightly faster than it will the two instruction, indexed form.

3.4.1 LINK

One of the intended features of the new LINK is to detect and remove unnecessary indirect references. That is, when one writes a program, or when a program is compiled, the author or compiler does not have the knowledge to predict which data references will be to data in the PC section and which will be to data in other sections. Therefore, the compilers will generate code that uses global (viz. indirect) references.

During the loading of the program, LINK will be able to determine whether a global reference is required and will be able to convert the indirect references into local references.

If the compilers are obligated to produce indexed references to data, this operation will become much more difficult, and in many cases impossible, to achieve.

4. Recommendation

4.1 Indirection

Ideally, the 2080 IBOX should be modified to detect and interpret instruction addressing of the form:

```
OP AC,@[EFIW BASE(X)]
```

That is, addressing using a single level of global indirection, possibly with indexing, should be handled completely and efficiently by the IBOX.

Many of the benefits of this change have been described already:

- "ideal" conformance to the architecture
- encourage the use of "apparent code"
- smaller programs
- allow current software development plans to proceed
- provide common, efficient code for KL-10 and 2080

4.2 paging cache

As stated earlier, more study is needed to know if we have a problem at all. However, it seems quite clear that some sort of microcode-supported cache in the EBOX is desirable, both for the 2080 and the KL-10, and should be included in the FRS 2080 microcode and be available for the "FCC" KL-10.

```
[Recording initiated at Wed 28-Jul-82 14:44:06]

LINK FROM HALL, TTY 205

 TOPS-20 Command processor 6(734)
@RUN KCMON

********
DECSYSTEM KC10 DIAGNOSTIC MONITOR
VERSION 11.0, SV=2.0, TOPS-20, KL10, CPU#=2123
WEDNESDAY, JULY 28, 82  14:44:16
********
KCMON>RUN DCKFB

DECSYSTEM KC10 INSTRUCTION TIMING TEST (DCKFB)
VERSION 0.1, SV=2.0, TOPS-20, KL10, CPU#=2123
WEDNESDAY, JULY 28, 82  14:44:22
SWITCHES = 000000 000000

        THE FOLLOWING TIMES ARE APPROXIMATIONS IN USER MODE
        DUE TO VARIOUS SYSTEM CONFLICTS.

37 - LUUO        =  1081 NSEC.

110 - DFAD       =  2146 NSEC.  (1 RIGHT SHIFT)
110 - DFAD       =  2144 NSEC.  (8 RIGHT SHIFT - 1 LEFT)
111 - DFSB       =  2383 NSEC.
112 - DFMP       =  4394 NSEC.
113 - DFDV       =  9026 NSEC.
114 - DADD       =  1133 NSEC.
115 - DSUB       =  1121 NSEC.
116 - DMUL       =  4727 NSEC.
117 - DDIV       = 10082 NSEC.

120 - DMOVE      =   768 NSEC.
121 - DMOVN      =  1010 NSEC.
122 - FIX        =   911 NSEC.  (A FLOATING POINT ONE)
124 - DMOVEM     =  1018 NSEC.
125 - DMOVNM     =  1257 NSEC.
126 - FIXR       =   920 NSEC.  (A FLOATING POINT ONE)
127 - FLTR       =  1610 NSEC.  (AN INTERGER ONE)

132 - FSC        =  1581 NSEC.  (AN INTERGER ONE)
133 - IBP        =   776 NSEC.  (INCREMENT POSITION)
133 - IBP        =   840 NSEC.  (INCREMENT Y)
133 - ADJBP      =  9671 NSEC.  (POSITIVE)
133 - ADJBP      =  9613 NSEC.  (NEGATIVE)
134 - ILDB       =  1393 NSEC.  (7 BITS)
134 - ILDB       =  1394 NSEC.  (6 BITS)
135 - LDB        =  1263 NSEC.  (7 BITS - POS 6)
135 - LDB        =  1258 NSEC.  (7 BITS - POS 13)
135 - LDB        =  1258 NSEC.  (6 BITS - POS 5)
135 - LDB        =  1271 NSEC.  (6 BITS - POS 11)
136 - IDPB       =  1806 NSEC.  (7 BITS)
136 - IDPB       =  1792 NSEC.  (6 BITS)
137 - DPB        =  1640 NSEC.  (7 BITS - POS 6)
137 - DPB        =  1577 NSEC.  (7 BITS - POS 13)
137 - DPB        =  1642 NSEC.  (6 BITS - POS 5)
137 - DPB        =  1569 NSEC.  (6 BITS - POS 11)

140 - FAD        =  1646 NSEC.  (1 RIGHT SHIFT)
140 - FAD        =  1680 NSEC.  (8 RIGHT SHIFT - 3 LEFT)
142 - FADM       =  1907 NSEC.
142 - FADB       =  1903 NSEC.
144 - FADR       =  1678 NSEC.
145 - FADRI      =  1610 NSEC.
```

user

7/28/82

2123

```
146 - FADRM      =  1940 NSEC.
147 - FADRB      =  1940 NSEC.

150 - FSB        =  2074 NSEC.
152 - FSBM       =  1944 NSEC.
153 - FSBB       =  1941 NSEC.
154 - FSBR       =  1991 NSEC.
155 - FSBRI      =  1920 NSEC.
156 - FSBRM      =  1880 NSEC.
157 - FSBRB      =  1872 NSEC.

160 - FMP        =  2440 NSEC.  (7 ADD/SUB - 14 SHIFTS)
162 - FMPM       =  2650 NSEC.
163 - FMPB       =  2647 NSEC.
164 - FMPR       =  2484 NSEC.
165 - FMPRI      =  2426 NSEC.
166 - FMPRM      =  2677 NSEC.
167 - FMPRB      =  2681 NSEC.

170 - FDV        =  4993 NSEC.
172 - FDVM       =  5197 NSEC.
173 - FDVB       =  5210 NSEC.
174 - FDVR       =  5019 NSEC.
175 - FDVRI      =  4970 NSEC.
176 - FDVRM      =  5240 NSEC.
177 - FDVRB      =  5277 NSEC.

200 - MOVE       =   420 NSEC.  (MOVE FROM MEMORY)
200 - MOVE       =   386 NSEC.  (MOVE FROM AC)
200 - MOVE       =   734 NSEC.  (MOVE INDIRECT)
200 - MOVE       =   419 NSEC.  (MOVE INDEXED)
201 - MOVEI      =   281 NSEC.
202 - MOVEM      =   595 NSEC.
203 - MOVES      =   735 NSEC.  (MOVES AC)
203 - MOVES      =   736 NSEC.  (MOVES MEMORY)
204 - MOVS       =   420 NSEC.
205 - MOVSI      =   280 NSEC.
206 - MOVSM      =   737 NSEC.
207 - MOVSS      =   736 NSEC.

210 - MOVN       =   596 NSEC.
211 - MOVNI      =   453 NSEC.
212 - MOVNM      =   910 NSEC.
213 - MOVNS      =   912 NSEC.
214 - MOVM       =   492 NSEC.  (NEGATIVE)
214 - MOVM       =   492 NSEC.  (POSITIVE)
215 - MOVMI      =   350 NSEC.
216 - MOVMM      =   805 NSEC.
217 - MOVMS      =   804 NSEC.

220 - IMUL       =  2269 NSEC.
221 - IMULI      =  2143 NSEC.
222 - IMULM      =  2458 NSEC.
223 - IMULB      =  2465 NSEC.
224 - MUL        =  2197 NSEC.  (9 ADD/SUB - 18 SHIFTS)
225 - MULI       =  2108 NSEC.
226 - MULM       =  2324 NSEC.
226 - MULB       =  2501 NSEC.

230 - IDIV       =  5219 NSEC.
231 - IDIVI      =  5111 NSEC.
232 - IDIVM      =  5223 NSEC.
233 - IDIVB      =  5418 NSEC.
234 - DIV        =  4870 NSEC.
235 - DIVI       =  4747 NSEC.
236 - DIVM       =  2782 NSEC.
```

```
237 - DIVB          =  2797 NSEC.

240 - ASH           =  1223 NSEC.  (POSITIVE 15)
240 - ASH           =  3268 NSEC.  (POSITIVE 15 OVERFLOW)
240 - ASH           =   735 NSEC.  (NEGATIVE 15)
241 - ROT           =   594 NSEC.  (LEFT 7)
241 - ROT           =   698 NSEC.  (RIGHT 7)
242 - LSH           =   560 NSEC.  (LEFT 35)
242 - LSH           =   665 NSEC.  (RIGHT 35)
243 - JFFO          =   905 NSEC.  (1B0)
243 - JFFO          =  2128 NSEC.  (1B35)
240 - ASHC          =  1498 NSEC.  (LEFT 15)
245 - ROTC          =   802 NSEC.  (LEFT 7)
246 - LSHC          =   984 NSEC.  (LEFT 71)
246 - LSHC          =  1013 NSEC.  (RIGHT 71)

250 - EXCH          =   559 NSEC.  (AC WITH AN AC)
250 - EXCH          =   736 NSEC.  (AC WITH MEMORY)
251 - BLT           =  1684 NSEC.  (MEMORY TO MEMORY 1 WORD)
251 - BLT           =  2247 NSEC.  (MEMORY TO MEMORY 2 WORDS)
251 - BLT           =  1640 NSEC.  (AC TO MEMORY 1 WORD)
251 - BLT           =  2199 NSEC.  (AC TO MEMORY 2 WORDS)
252 - AOBJP         =   458 NSEC.
253 - AOBJN         =   460 NSEC.
254 - JRST          =   315 NSEC.
254 - JRSTF         =   876 NSEC.
255 - JFCL          =   765 NSEC.
256 - XCT           =   495 NSEC.

260 - PUSHJ         =  1012 NSEC.
261 - PUSH          =  1010 NSEC.
262 - POP           =   761 NSEC.
264 - JSR           =   679 NSEC.
265 - JSP           =   386 NSEC.
266 - JSA           =   640 NSEC.
267 - JRA           =   770 NSEC.

270 - ADD           =   422 NSEC.
271 - ADDI          =   320 NSEC.
272 - ADDM          =   777 NSEC.
273 - ADDB          =   771 NSEC.
274 - SUB           =   525 NSEC.
275 - SUBI          =   425 NSEC.
276 - SUBM          =   881 NSEC.
277 - SUBB          =   739 NSEC.

300 - CAI           =   419 NSEC.
301 - CAIL          =   420 NSEC.
302 - CAIE          =   422 NSEC.
303 - CAILE         =   424 NSEC.
304 - CAIA          =   421 NSEC.
305 - CAIGE         =   420 NSEC.
306 - CAIN          =   423 NSEC.
307 - CAIG          =   420 NSEC.

310 - CAM           =   559 NSEC.
311 - CAML          =   567 NSEC.
312 - CAME          =   572 NSEC.
313 - CAMLE         =   572 NSEC.
314 - CAMA          =   561 NSEC.
315 - CAMGE         =   573 NSEC.
316 - CAMN          =   569 NSEC.
317 - CAMG          =   565 NSEC.

320 - JUMP          =   420 NSEC.
321 - JUMPL         =   419 NSEC.
```

322 - JUMPE = 420 NSEC.
323 - JUMPLE = 420 NSEC.
324 - JUMPA = 420 NSEC.
325 - JUMPGE = 421 NSEC.
326 - JUMPN = 422 NSEC.
327 - JUMPG = 419 NSEC.

330 - SKIP = 559 NSEC.
331 - SKIPL = 567 NSEC.
332 - SKIPE = 583 NSEC.
333 - SKIPLE = 572 NSEC.
334 - SKIPA = 563 NSEC.
335 - SKIPGE = 574 NSEC.
336 - SKIPN = 567 NSEC.
337 - SKIPG = 566 NSEC.

340 - AOJ = 455 NSEC.
341 - AOJL = 456 NSEC.
342 - AOJE = 455 NSEC.
343 - AOJLE = 453 NSEC.
344 - AOJA = 455 NSEC.
345 - AOJGE = 456 NSEC.
346 - AOJN = 455 NSEC.
347 - AOJG = 454 NSEC.

350 - AOS = 734 NSEC.
351 - AOSL = 748 NSEC.
352 - AOSE = 745 NSEC.
353 - AOSLE = 746 NSEC.
354 - AOSA = 745 NSEC.
355 - AOSGE = 748 NSEC.
356 - AOSN = 753 NSEC.
357 - AOSG = 746 NSEC.

360 - SOJ = 564 NSEC.
361 - SOJL = 559 NSEC.
362 - SOJE = 562 NSEC.
363 - SOJLE = 559 NSEC.
364 - SOJA = 841 NSEC.
365 - SOJGE = 559 NSEC.
366 - SOJN = 560 NSEC.
367 - SOJG = 558 NSEC.

370 - SOS = 752 NSEC.
371 - SOSL = 745 NSEC.
372 - SOSE = 743 NSEC.
373 - SOSLE = 746 NSEC.
374 - SOSA = 647 NSEC.
375 - SOSGE = 743 NSEC.
376 - SOSN = 747 NSEC.
377 - SOSG = 740 NSEC.

400 - SETZ = 281 NSEC.
401 - SETZI = 280 NSEC.
402 - SETZM = 595 NSEC.
403 - SETZB = 595 NSEC.
404 - AND = 423 NSEC.
405 - ANDI = 315 NSEC.
406 - ANDM = 770 NSEC.
407 - ANDB = 771 NSEC.

410 - ANDCA = 421 NSEC.
411 - ANDCAI = 315 NSEC.
412 - ANDCAM = 770 NSEC.
413 - ANDCAB = 770 NSEC.
414 - SETM = 385 NSEC.

415 - SETMI = 351 NSEC.
416 - SETMM = 735 NSEC.
417 - SETMB = 736 NSEC.

420 - ANDCM = 423 NSEC.
421 - ANDCMI = 315 NSEC.
422 - ANDCMM = 767 NSEC.
423 - ANDCMB = 771 NSEC.
424 - SETA = 385 NSEC.
425 - SETAI = 281 NSEC.
426 - SETAM = 733 NSEC.
427 - SETAB = 737 NSEC.

430 - XOR = 420 NSEC.
431 - XORI = 316 NSEC.
432 - XORM = 768 NSEC.
433 - XORB = 773 NSEC.
434 - OR = 417 NSEC.
435 - ORI = 315 NSEC.
436 - ORM = 769 NSEC.
437 - ORB = 769 NSEC.

440 - ANDCB = 420 NSEC.
441 - ANDCBI = 315 NSEC.
442 - ANDCBM = 772 NSEC.
443 - ANDCBB = 768 NSEC.
444 - EQV = 420 NSEC.
445 - EQVI = 315 NSEC.
446 - EQVM = 771 NSEC.
447 - EQVB = 768 NSEC.

450 - SETCA = 319 NSEC.
451 - SETCAI = 315 NSEC.
452 - SETCAM = 633 NSEC.
453 - SETCAB = 630 NSEC.
454 - ORCA = 420 NSEC.
455 - ORCAI = 315 NSEC.
456 - ORCAM = 771 NSEC.
457 - ORCAB = 768 NSEC.

460 - SETCM = 385 NSEC.
461 - SETCMI = 280 NSEC.
462 - SETCMM = 736 NSEC.
463 - SETCMB = 735 NSEC.
464 - ORCM = 419 NSEC.
465 - ORCMI = 315 NSEC.
466 - ORCMM = 770 NSEC.
467 - ORCMB = 769 NSEC.

470 - ORCB = 419 NSEC.
471 - ORCBI = 316 NSEC.
472 - ORCBM = 771 NSEC.
473 - ORCBB = 766 NSEC.
474 - SETO = 281 NSEC.
475 - SETOI = 280 NSEC.
476 - SETOM = 596 NSEC.
477 - SETOB = 594 NSEC.

500 - HLL = 422 NSEC.
501 - HLLI = 386 NSEC.
502 - HLLM = 769 NSEC.
503 - HLLS = 733 NSEC.
504 - HRL = 420 NSEC.
505 - HRLI = 316 NSEC.
506 - HRLM = 840 NSEC.
507 - HRLS = 734 NSEC.

510 - HLLZ = 384 NSEC.
511 - HLLZI = 283 NSEC.
512 - HLLZM = 734 NSEC.
513 - HLLZS = 736 NSEC.
514 - HRLZ = 385 NSEC.
515 - HRLZI = 282 NSEC.
516 - HRLZM = 735 NSEC.
517 - HRLZS = 734 NSEC.

520 - HLLO = 385 NSEC.
521 - HLLOI = 281 NSEC.
522 - HLLOM = 734 NSEC.
523 - HLLOS = 735 NSEC.
524 - HRLO = 385 NSEC.
525 - HRLOI = 281 NSEC.
526 - HRLOM = 732 NSEC.
527 - HRLOS = 736 NSEC.

530 - HLLE = 458 NSEC.
531 - HLLEI = 352 NSEC.
532 - HLLEM = 803 NSEC.
533 - HLLES = 805 NSEC.
534 - HRLE = 455 NSEC.
535 - HRLEI = 350 NSEC.
536 - HRLEM = 802 NSEC.
537 - HRLES = 804 NSEC.

540 - HRR = 420 NSEC.
541 - HRRI = 315 NSEC.
542 - HRRM = 767 NSEC.
543 - HRRS = 734 NSEC.
544 - HLR = 422 NSEC.
545 - HLRI = 316 NSEC.
546 - HLRM = 839 NSEC.
547 - HLRS = 735 NSEC.

550 - HRRZ = 386 NSEC.
551 - HRRZI = 281 NSEC.
552 - HRRZM = 734 NSEC.
553 - HRRZS = 733 NSEC.
554 - HLRZ = 386 NSEC.
555 - HLRZI = 281 NSEC.
556 - HLRZM = 734 NSEC.
557 - HLRZS = 735 NSEC.

560 - HRRO = 389 NSEC.
561 - HRROI = 280 NSEC.
562 - HRROM = 735 NSEC.
563 - HRROS = 736 NSEC.
564 - HLRO = 385 NSEC.
565 - HLROI = 280 NSEC.
566 - HLROM = 739 NSEC.
567 - HLROS = 735 NSEC.

570 - HRRE = 454 NSEC.
571 - HRREI = 350 NSEC.
572 - HRREM = 802 NSEC.
573 - HRRES = 808 NSEC.
574 - HLRE = 454 NSEC.
575 - HLREI = 350 NSEC.
576 - HLREM = 802 NSEC.
577 - HLRES = 806 NSEC.

600 - TRN = 280 NSEC.
601 - TLN = 281 NSEC.

602 - TRNE = 422 NSEC.
603 - TLNE = 493 NSEC.
604 - TRNA = 386 NSEC.
605 - TLNA = 387 NSEC.
606 - TRNN = 420 NSEC.
607 - TLNN = 491 NSEC.

610 - TDN = 283 NSEC.
611 - TSN = 281 NSEC.
612 - TDNE = 563 NSEC.
613 - TSNE = 761 NSEC.
614 - TDNA = 538 NSEC.
615 - TSNA = 529 NSEC.
616 - TDNN = 558 NSEC.
617 - TSNN = 628 NSEC.

620 - TRZ = 385 NSEC.
621 - TLZ = 386 NSEC.
622 - TRZE = 423 NSEC.
623 - TLZE = 492 NSEC.
624 - TRZA = 386 NSEC.
625 - TLZA = 387 NSEC.
626 - TRZN = 421 NSEC.
627 - TLZN = 488 NSEC.

630 - TDZ = 523 NSEC.
631 - TSZ = 524 NSEC.
632 - TDZE = 566 NSEC.
633 - TSZE = 636 NSEC.
634 - TDZA = 527 NSEC.
635 - TSZA = 529 NSEC.
636 - TDZN = 560 NSEC.
637 - TSZN = 632 NSEC.

640 - TRC = 390 NSEC.
641 - TLC = 385 NSEC.
642 - TRCE = 426 NSEC.
643 - TLCE = 492 NSEC.
644 - TRCA = 387 NSEC.
645 - TLCA = 387 NSEC.
646 - TRCN = 421 NSEC.
647 - TLCN = 490 NSEC.

650 - TDC = 524 NSEC.
651 - TSC = 524 NSEC.
652 - TDCE = 565 NSEC.
653 - TSCE = 634 NSEC.
654 - TDCA = 530 NSEC.
655 - TSCA = 530 NSEC.
656 - TDCN = 561 NSEC.
657 - TSCN = 631 NSEC.

660 - TRO = 384 NSEC.
661 - TLO = 385 NSEC.
662 - TROE = 422 NSEC.
663 - TLOE = 493 NSEC.
664 - TROA = 388 NSEC.
665 - TLOA = 390 NSEC.
666 - TRON = 418 NSEC.
667 - TLON = 489 NSEC.

670 - TDO = 528 NSEC.
671 - TSO = 525 NSEC.
672 - TDOE = 565 NSEC.
673 - TSOE = 631 NSEC.
674 - TDOA = 528 NSEC.

```
675 - TSOA        =    532 NSEC.
676 - TDON        =    560 NSEC.
677 - TSON        =    627 NSEC.

123 - EXTEND      =   4767 NSEC.  (OVERHEAD - MOVSI - BLT (6 WDS))

001 - CMPSL       =   5408 NSEC.  (1 BYTE)
001 - CMPSL       =   7510 NSEC.  (2 BYTES)
001 - CMPSL       =  14602 NSEC.  (5 BYTES)

002 - CMPSE       =   5338 NSEC.  (1 BYTE)
002 - CMPSE       =   7700 NSEC.  (2 BYTES)
002 - CMPSE       =  14830 NSEC.  (5 BYTES)

003 - CMPSLE      =   5262 NSEC.  (1 BYTE)
003 - CMPSLE      =   7547 NSEC.  (2 BYTES)
003 - CMPSLE      =  14726 NSEC.  (5 BYTES)

004 - EDIT        = 101810 NSEC.  (BLANK)
004 - EDIT        =  68159 NSEC.  ($.01 DUE US)
004 - EDIT        =  66210 NSEC.  ($99999.99 DUE US)
004 - EDIT        =  66165 NSEC.  ($99999.99 CREDIT)

005 - CMPSGE      =   5376 NSEC.  (1 BYTE)
005 - CMPSGE      =   7526 NSEC.  (2 BYTES)
005 - CMPSGE      =  14785 NSEC.  (5 BYTES)

006 - CMPSN       =   5278 NSEC.  (1 BYTE)
006 - CMPSN       =   7582 NSEC.  (2 BYTES)
006 - CMPSN       =  14877 NSEC.  (5 BYTES)

007 - CMPSG       =   5249 NSEC.  (1 BYTE)
007 - CMPSG       =   7510 NSEC.  (2 BYTES)
007 - CMPSG       =  14706 NSEC.  (5 BYTES)

010 - CVTDBO      =   5170 NSEC.  (1 BYTE)
010 - CVTDBO      =   7009 NSEC.  (2 BYTES)
010 - CVTDBO      =  11810 NSEC.  (5 BYTES)

011 - CVTDBT      =   5590 NSEC.  (1 BYTE)
011 - CVTDBT      =   7764 NSEC.  (2 BYTES)
011 - CVTDBT      =  14248 NSEC.  (5 BYTES)

012 - CVTBDO      =  14725 NSEC.  (1 BYTE)
012 - CVTBDO      =  16781 NSEC.  (2 BYTES)
012 - CVTBDO      =  24081 NSEC.  (5 BYTES)

013 - CVTBDT      =  14813 NSEC.  (1 BYTE)
013 - CVTBDT      =  17452 NSEC.  (2 BYTE)
013 - CVTBDT      =  25845 NSEC.  (5 BYTE)

014 - MOVSO       =   6812 NSEC.  (1 BYTE)
014 - MOVSO       =   9300 NSEC.  (2 BYTES)
014 - MOVSO       =  17300 NSEC.  (5 BYTES)

015 - MOVST       =   7123 NSEC.  (1 BYTE)
015 - MOVST       =  10425 NSEC.  (2 BYTES)
015 - MOVST       =  19322 NSEC.  (5 BYTES)

016 - MOVSLJ      =   5787 NSEC.  (1 BYTE)
016 - MOVSLJ      =   7917 NSEC.  (2 BYTES)
016 - MOVSLJ      =  14864 NSEC.  (5 BYTES)

017 - MOVSRJ      =   6521 NSEC.  (1 BYTE)
017 - MOVSRJ      =   8560 NSEC.  (2 BYTES)
017 - MOVSRJ      =  15360 NSEC.  (5 BYTES)
```

```
   STD
>,
PDP-10 KL10 SPECIAL INSTRUCTION TIMING TEST (DLKFB)
VERSION 0.1, SV=0.1, CPU#=2123, MCV=275, MCO=40, HO=36, 60HZ

SWITCHES = 000000 000000
CLK SOURCE = EXTERN, CLK RATE = FULL, AC BLK 0 , CACHE: 0 1 2 3

CLOCK CYCLE       =    33 NSEC.
INDEXING          =    34 NSEC.
INDIRECT          =   237 NSEC.

37 - LUUO         =  1061 NSEC.

40 - MUUO         =  4381 NSEC.

110 - DFAD        =  2100 NSEC.  (1 RIGHT SHIFT)
110 - DFAD        =  2100 NSEC.  (8 RIGHT SHIFT - 1 LEFT)
111 - DFSB        =  2351 NSEC.
112 - DFMP        =  4456 NSEC.
113 - DFDV        =  9625 NSEC.
114 - DADD        =  1091 NSEC.
115 - DSUB        =  1088 NSEC.
116 - DMUL        =  4783 NSEC.
117 - DDIV        = 10427 NSEC.

120 - DMOVE       =   745 NSEC.
121 - DMOVN       =   986 NSEC.
122 - FIX         =   878 NSEC.  (A FLOATING POINT ONE)
124 - DMOVEM      =   986 NSEC.
125 - DMOVNM      =  1230 NSEC.
126 - FIXR        =   878 NSEC.  (A FLOATING POINT ONE)
127 - FLTR        =  1568 NSEC.  (AN INTERGER ONE)

132 - FSC         =  1533 NSEC.  (AN INTERGER ONE)
133 - IBP         =   681 NSEC.  (INCREMENT POSITION)
133 - IBP         =   749 NSEC.  (INCREMENT Y)
133 - ADJBP       =  9308 NSEC.  (POSITIVE)
133 - ADJBP       =  9308 NSEC.  (NEGATIVE)
134 - ILDB        =  1121 NSEC.  (7 BITS)
134 - ILDB        =  1121 NSEC.  (6 BITS)
135 - LDB         =   986 NSEC.  (7 BITS - POS 6)
135 - LDB         =   986 NSEC.  (7 BITS - POS 13)
135 - LDB         =   986 NSEC.  (6 BITS - POS 5)
135 - LDB         =   986 NSEC.  (6 BITS - POS 11)
136 - IDPB        =  1532 NSEC.  (7 BITS)
136 - IDPB        =  1532 NSEC.  (6 BITS)
137 - DPB         =  1369 NSEC.  (7 BITS - POS 6)
137 - DPB         =  1300 NSEC.  (7 BITS - POS 13)
137 - DPB         =  1369 NSEC.  (6 BITS - POS 5)
137 - DPB         =  1300 NSEC.  (6 BITS - POS 11)

140 - FAD         =  1601 NSEC.  (1 RIGHT SHIFT)
140 - FAD         =  1601 NSEC.  (8 RIGHT SHIFT - 3 LEFT)
142 - FADM        =  1842 NSEC.
142 - FADB        =  1842 NSEC.
144 - FADR        =  1636 NSEC.
145 - FADRI       =  1568 NSEC.
146 - FADRM       =  1878 NSEC.
147 - FADRB       =  1878 NSEC.

150 - FSB         =  2017 NSEC.
152 - FSBM        =  1874 NSEC.
153 - FSBB        =  1878 NSEC.
154 - FSBR        =  1947 NSEC.
155 - FSBRI       =  1878 NSEC.
```

7/28/82
2123
exec

```
156 - FSBRM       =  1808 NSEC.
157 - FSBRB       =  1808 NSEC.

160 - FMP         =  2410 NSEC.  (7 ADD/SUB - 14 SHIFTS)
162 - FMPM        =  2570 NSEC.
163 - FMPB        =  2570 NSEC.
164 - FMPR        =  2440 NSEC.
165 - FMPRI       =  2370 NSEC.
166 - FMPRM       =  2609 NSEC.
167 - FMPRB       =  2609 NSEC.

170 - FDV         =  5052 NSEC.
172 - FDVM        =  5165 NSEC.
173 - FDVB        =  5165 NSEC.
174 - FDVR        =  5068 NSEC.
175 - FDVRI       =  5000 NSEC.
176 - FDVRM       =  5204 NSEC.
177 - FDVRB       =  5204 NSEC.

200 - MOVE        =   403 NSEC.  (MOVE FROM MEMORY)
200 - MOVE        =   369 NSEC.  (MOVE FROM AC)
200 - MOVE        =   676 NSEC.  (MOVE INDIRECT)
200 - MOVE        =   403 NSEC.  (MOVE INDEXED)
201 - MOVEI       =   268 NSEC.
202 - MOVEM       =   573 NSEC.
203 - MOVES       =   710 NSEC.  (MOVES AC)
203 - MOVES       =   710 NSEC.  (MOVES MEMORY)
204 - MOVS        =   403 NSEC.
205 - MOVSI       =   268 NSEC.
206 - MOVSM       =   710 NSEC.
207 - MOVSS       =   710 NSEC.

210 - MOVN        =   573 NSEC.
211 - MOVNI       =   437 NSEC.
212 - MOVNM       =   883 NSEC.
213 - MOVNS       =   883 NSEC.
214 - MOVM        =   471 NSEC.  (NEGATIVE)
214 - MOVM        =   471 NSEC.  (POSITIVE)
215 - MOVMI       =   336 NSEC.
216 - MOVMM       =   779 NSEC.
217 - MOVMS       =   779 NSEC.

220 - IMUL        =  2226 NSEC.
221 - IMULI       =  2087 NSEC.
222 - IMULM       =  2397 NSEC.
223 - IMULB       =  2397 NSEC.
224 - MUL         =  2162 NSEC.  (9 ADD/SUB - 18 SHIFTS)
225 - MULI        =  2052 NSEC.
226 - MULM        =  2256 NSEC.
226 - MULB        =  2431 NSEC.

230 - IDIV        =  5310 NSEC.
231 - IDIVI       =  5163 NSEC.
232 - IDIVM       =  5204 NSEC.
233 - IDIVB       =  5387 NSEC.
234 - DIV         =  4911 NSEC.
235 - DIVI        =  4763 NSEC.
236 - DIVM        =  1910 NSEC.
237 - DIVB        =  1910 NSEC.

240 - ASH         =  1188 NSEC.  (POSITIVE 15)
240 - ASH         =  2331 NSEC.  (POSITIVE 15 OVERFLOW)
240 - ASH         =   708 NSEC.  (NEGATIVE 15)
241 - ROT         =   573 NSEC.  (LEFT 7)
241 - ROT         =   676 NSEC.  (RIGHT 7)
242 - LSH         =   573 NSEC.  (LEFT 7)
```

```
242 - LSH       =   641 NSEC.   (RIGHT 35)
243 - JFFO      =   885 NSEC.   (1B0)
243 - JFFO      =  2096 NSEC.   (1B35)
240 - ASHC      =  1462 NSEC.   (LEFT 15)
245 - ROTC      =   779 NSEC.   (LEFT 7)
246 - LSHC      =   952 NSEC.   (LEFT 71)
246 - LSHC      =   986 NSEC.   (RIGHT 71)

250 - EXCH      =   539 NSEC.   (AC WITH AN AC)
250 - EXCH      =   710 NSEC.   (AC WITH MEMORY)
251 - BLT       =  1640 NSEC.   (MEMORY TO MEMORY 1 WORD)
251 - BLT       =  2196 NSEC.   (MEMORY TO MEMORY 2 WORDS)
251 - BLT       =  1606 NSEC.   (AC TO MEMORY 1 WORD)
251 - BLT       =  2127 NSEC.   (AC TO MEMORY 2 WORDS)
252 - AOBJP     =   437 NSEC.
253 - AOBJN     =   437 NSEC.
254 - JRST      =   302 NSEC.
254 - JRSTF     =   807 NSEC.
255 - JFCL      =   745 NSEC.
256 - XCT       =   555 NSEC.

260 - PUSHJ     =   976 NSEC.
261 - PUSH      =   976 NSEC.
262 - POP       =   743 NSEC.
264 - JSR       =   570 NSEC.
265 - JSP       =   369 NSEC.
266 - JSA       =   604 NSEC.
267 - JRA       =   675 NSEC.

270 - ADD       =   403 NSEC.
271 - ADDI      =   302 NSEC.
272 - ADDM      =   743 NSEC.
273 - ADDB      =   743 NSEC.
274 - SUB       =   505 NSEC.
275 - SUBI      =   403 NSEC.
276 - SUBM      =   845 NSEC.
277 - SUBB      =   708 NSEC.

300 - CAI       =   403 NSEC.
301 - CAIL      =   403 NSEC.
302 - CAIE      =   403 NSEC.
303 - CAILE     =   403 NSEC.
304 - CAIA      =   402 NSEC.
305 - CAIGE     =   403 NSEC.
306 - CAIN      =   402 NSEC.
307 - CAIG      =   403 NSEC.

310 - CAM       =   539 NSEC.
311 - CAML      =   539 NSEC.
312 - CAME      =   536 NSEC.
313 - CAMLE     =   536 NSEC.
314 - CAMA      =   536 NSEC.
315 - CAMGE     =   536 NSEC.
316 - CAMN      =   540 NSEC.
317 - CAMG      =   540 NSEC.

320 - JUMP      =   403 NSEC.
321 - JUMPL     =   403 NSEC.
322 - JUMPE     =   403 NSEC.
323 - JUMPLE    =   403 NSEC.
324 - JUMPA     =   403 NSEC.
325 - JUMPGE    =   403 NSEC.
326 - JUMPN     =   403 NSEC.
327 - JUMPG     =   403 NSEC.
```

331 - SKIPL = 540 NSEC.
332 - SKIPE = 536 NSEC.
333 - SKIPLE = 536 NSEC.
334 - SKIPA = 536 NSEC.
335 - SKIPGE = 536 NSEC.
336 - SKIPN = 539 NSEC.
337 - SKIPG = 539 NSEC.

340 - AOJ = 437 NSEC.
341 - AOJL = 437 NSEC.
342 - AOJE = 437 NSEC.
343 - AOJLE = 437 NSEC.
344 - AOJA = 437 NSEC.
345 - AOJGE = 437 NSEC.
346 - AOJN = 437 NSEC.
347 - AOJG = 437 NSEC.

350 - AOS = 710 NSEC.
351 - AOSL = 711 NSEC.
352 - AOSE = 711 NSEC.
353 - AOSLE = 711 NSEC.
354 - AOSA = 705 NSEC.
355 - AOSGE = 705 NSEC.
356 - AOSN = 705 NSEC.
357 - AOSG = 705 NSEC.

360 - SOJ = 539 NSEC.
361 - SOJL = 539 NSEC.
362 - SOJE = 539 NSEC.
363 - SOJLE = 539 NSEC.
364 - SOJA = 807 NSEC.
365 - SOJGE = 539 NSEC.
366 - SOJN = 539 NSEC.
367 - SOJG = 539 NSEC.

370 - SOS = 711 NSEC.
371 - SOSL = 705 NSEC.
372 - SOSE = 711 NSEC.
373 - SOSLE = 705 NSEC.
374 - SOSA = 603 NSEC.
375 - SOSGE = 711 NSEC.
376 - SOSN = 705 NSEC.
377 - SOSG = 711 NSEC.

400 - SETZ = 268 NSEC.
401 - SETZI = 268 NSEC.
402 - SETZM = 573 NSEC.
403 - SETZB = 573 NSEC.
404 - AND = 403 NSEC.
405 - ANDI = 302 NSEC.
406 - ANDM = 745 NSEC.
407 - ANDB = 745 NSEC.

410 - ANDCA = 403 NSEC.
411 - ANDCAI = 302 NSEC.
412 - ANDCAM = 745 NSEC.
413 - ANDCAB = 745 NSEC.
414 - SETM = 369 NSEC.
415 - SETMI = 268 NSEC.
416 - SETMM = 710 NSEC.
417 - SETMB = 710 NSEC.

420 - ANDCM = 403 NSEC.
421 - ANDCMI = 302 NSEC.
422 - ANDCMM = 745 NSEC.
423 - ANDCMB = 745 NSEC.

424 - SETA = 369 NSEC.
425 - SETAI = 268 NSEC.
426 - SETAM = 710 NSEC.
427 - SETAB = 710 NSEC.

430 - XOR = 403 NSEC.
431 - XORI = 302 NSEC.
432 - XORM = 745 NSEC.
433 - XORB = 744 NSEC.
434 - OR = 403 NSEC.
435 - ORI = 302 NSEC.
436 - ORM = 745 NSEC.
437 - ORB = 744 NSEC.

440 - ANDCB = 403 NSEC.
441 - ANDCBI = 302 NSEC.
442 - ANDCBM = 745 NSEC.
443 - ANDCBB = 745 NSEC.
444 - EQV = 403 NSEC.
445 - EQVI = 302 NSEC.
446 - EQVM = 745 NSEC.
447 - EQVB = 745 NSEC.

450 - SETCA = 302 NSEC.
451 - SETCAI = 302 NSEC.
452 - SETCAM = 607 NSEC.
453 - SETCAB = 607 NSEC.
454 - ORCA = 403 NSEC.
455 - ORCAI = 302 NSEC.
456 - ORCAM = 745 NSEC.
457 - ORCAB = 745 NSEC.

460 - SETCM = 369 NSEC.
461 - SETCMI = 268 NSEC.
462 - SETCMM = 710 NSEC.
463 - SETCMB = 710 NSEC.
464 - ORCM = 403 NSEC.
465 - ORCMI = 302 NSEC.
466 - ORCMM = 745 NSEC.
467 - ORCMB = 745 NSEC.

470 - ORCB = 403 NSEC.
471 - ORCBI = 302 NSEC.
472 - ORCBM = 745 NSEC.
473 - ORCBB = 745 NSEC.
474 - SETO = 268 NSEC.
475 - SETOI = 268 NSEC.
476 - SETOM = 573 NSEC.
477 - SETOB = 573 NSEC.

500 - HLL = 403 NSEC.
501 - HLLI = 302 NSEC.
502 - HLLM = 745 NSEC.
503 - HLLS = 710 NSEC.
504 - HRL = 403 NSEC.
505 - HRLI = 302 NSEC.
506 - HRLM = 813 NSEC.
507 - HRLS = 710 NSEC.

510 - HLLZ = 369 NSEC.
511 - HLLZI = 268 NSEC.
512 - HLLZM = 710 NSEC.
513 - HLLZS = 710 NSEC.
514 - HRLZ = 369 NSEC.
515 - HRLZI = 268 NSEC.

517 - HRLZS = 710 NSEC.

520 - HLLO = 369 NSEC.
521 - HLLOI = 268 NSEC.
522 - HLLOM = 710 NSEC.
523 - HLLOS = 710 NSEC.
524 - HRLO = 369 NSEC.
525 - HRLOI = 268 NSEC.
526 - HRLOM = 710 NSEC.
527 - HRLOS = 710 NSEC.

530 - HLLE = 437 NSEC.
531 - HLLEI = 335 NSEC.
532 - HLLEM = 779 NSEC.
533 - HLLES = 779 NSEC.
534 - HRLE = 437 NSEC.
535 - HRLEI = 335 NSEC.
536 - HRLEM = 779 NSEC.
537 - HRLES = 779 NSEC.

540 - HRR = 403 NSEC.
541 - HRRI = 302 NSEC.
542 - HRRM = 744 NSEC.
543 - HRRS = 710 NSEC.
544 - HLR = 403 NSEC.
545 - HLRI = 302 NSEC.
546 - HLRM = 813 NSEC.
547 - HLRS = 710 NSEC.

550 - HRRZ = 369 NSEC.
551 - HRRZI = 268 NSEC.
552 - HRRZM = 710 NSEC.
553 - HRRZS = 710 NSEC.
554 - HLRZ = 369 NSEC.
555 - HLRZI = 268 NSEC.
556 - HLRZM = 710 NSEC.
557 - HLRZS = 710 NSEC.

560 - HRRO = 369 NSEC.
561 - HRROI = 268 NSEC.
562 - HRROM = 710 NSEC.
563 - HRROS = 710 NSEC.
564 - HLRO = 369 NSEC.
565 - HLROI = 268 NSEC.
566 - HLROM = 710 NSEC.
567 - HLROS = 710 NSEC.

570 - HRRE = 437 NSEC.
571 - HRREI = 335 NSEC.
572 - HRREM = 779 NSEC.
573 - HRRES = 779 NSEC.
574 - HLRE = 437 NSEC.
575 - HLREI = 335 NSEC.
576 - HLREM = 779 NSEC.
577 - HLRES = 779 NSEC.

600 - TRN = 268 NSEC.
601 - TLN = 268 NSEC.
602 - TRNE = 402 NSEC.
603 - TLNE = 469 NSEC.
604 - TRNA = 368 NSEC.
605 - TLNA = 368 NSEC.
606 - TRNN = 403 NSEC.
607 - TLNN = 471 NSEC.

610 - TDN = 268 NSEC.

611 - TSN = 268 NSEC.
612 - TDNE = 536 NSEC.
613 - TSNE = 603 NSEC.
614 - TDNA = 502 NSEC.
615 - TSNA = 502 NSEC.
616 - TDNN = 539 NSEC.
617 - TSNN = 608 NSEC.

620 - TRZ = 369 NSEC.
621 - TLZ = 369 NSEC.
622 - TRZE = 402 NSEC.
623 - TLZE = 469 NSEC.
624 - TRZA = 368 NSEC.
625 - TLZA = 368 NSEC.
626 - TRZN = 403 NSEC.
627 - TLZN = 471 NSEC.

630 - TDZ = 505 NSEC.
631 - TSZ = 505 NSEC.
632 - TDZE = 536 NSEC.
633 - TSZE = 603 NSEC.
634 - TDZA = 502 NSEC.
635 - TSZA = 502 NSEC.
636 - TDZN = 539 NSEC.
637 - TSZN = 608 NSEC.

640 - TRC = 369 NSEC.
641 - TLC = 369 NSEC.
642 - TRCE = 402 NSEC.
643 - TLCE = 469 NSEC.
644 - TRCA = 368 NSEC.
645 - TLCA = 368 NSEC.
646 - TRCN = 403 NSEC.
647 - TLCN = 471 NSEC.

650 - TDC = 505 NSEC.
651 - TSC = 505 NSEC.
652 - TDCE = 536 NSEC.
653 - TSCE = 603 NSEC.
654 - TDCA = 502 NSEC.
655 - TSCA = 502 NSEC.
656 - TDCN = 539 NSEC.
657 - TSCN = 608 NSEC.

660 - TRO = 369 NSEC.
661 - TLO = 369 NSEC.
662 - TROE = 402 NSEC.
663 - TLOE = 469 NSEC.
664 - TROA = 368 NSEC.
665 - TLOA = 368 NSEC.
666 - TRON = 403 NSEC.
667 - TLON = 471 NSEC.

670 - TDO = 505 NSEC.
671 - TSO = 505 NSEC.
672 - TDOE = 536 NSEC.
673 - TSOE = 603 NSEC.
674 - TDOA = 502 NSEC.
675 - TSOA = 502 NSEC.
676 - TDON = 539 NSEC.
677 - TSON = 608 NSEC.

123 - EXTEND = 4759 NSEC. (OVERHEAD - MOVSI - BLT (6 WDS))

001 - CMPSL = 4268 NSEC. (1 BYTE)

```
001 - CMPSL       = 14957 NSEC.  (5 BYTES)

002 - CMPSE       =  4205 NSEC.  (1 BYTE)
002 - CMPSE       =  6604 NSEC.  (2 BYTES)
002 - CMPSE       = 14519 NSEC.  (5 BYTES)

003 - CMPSLE      =  4140 NSEC.  (1 BYTE)
003 - CMPSLE      =  6551 NSEC.  (2 BYTES)
003 - CMPSLE      = 14370 NSEC.  (5 BYTES)

004 - EDIT        = 188291 NSEC. (BLANK)
004 - EDIT        = 94723 NSEC.  ($.01 DUE US)
004 - EDIT        = 90861 NSEC.  ($99999.99 DUE US)
004 - EDIT        = 90861 NSEC.  ($99999.99 CREDIT)

005 - CMPSGE      =  4140 NSEC.  (1 BYTE)
005 - CMPSGE      =  6551 NSEC.  (2 BYTES)
005 - CMPSGE      = 14370 NSEC.  (5 BYTES)

006 - CMPSN       =  4357 NSEC.  (1 BYTE)
006 - CMPSN       =  6811 NSEC.  (2 BYTES)
006 - CMPSN       = 15169 NSEC.  (5 BYTES)

007 - CMPSG       =  4268 NSEC.  (1 BYTE)
007 - CMPSG       =  6739 NSEC.  (2 BYTES)
007 - CMPSG       = 14957 NSEC.  (5 BYTES)

010 - CVTDBO      =  4439 NSEC.  (1 BYTE)
010 - CVTDBO      =  6151 NSEC.  (2 BYTES)
010 - CVTDBO      = 11578 NSEC.  (5 BYTES)

011 - CVTDBT      =  4947 NSEC.  (1 BYTE)
011 - CVTDBT      =  7211 NSEC.  (2 BYTES)
011 - CVTDBT      = 14370 NSEC.  (5 BYTES)

012 - CVTBDO      = 14672 NSEC.  (1 BYTE)
012 - CVTBDO      = 17280 NSEC.  (2 BYTES)
012 - CVTBDO      = 26122 NSEC.  (5 BYTES)

013 - CVTBDT      = 14983 NSEC.  (1 BYTE)
013 - CVTBDT      = 18098 NSEC.  (2 BYTE)
013 - CVTBDT      = 28187 NSEC.  (5 BYTE)

014 - MOVSO       =  5600 NSEC.  (1 BYTE)
014 - MOVSO       =  8289 NSEC.  (2 BYTES)
014 - MOVSO       = 17280 NSEC.  (5 BYTES)

015 - MOVST       =  6008 NSEC.  (1 BYTE)
015 - MOVST       =  9254 NSEC.  (2 BYTES)
015 - MOVST       = 19686 NSEC.  (5 BYTES)

016 - MOVSLJ      =  4579 NSEC.  (1 BYTE)
016 - MOVSLJ      =  6817 NSEC.  (2 BYTES)
016 - MOVSLJ      = 14370 NSEC.  (5 BYTES)

017 - MOVSRJ      =  5182 NSEC.  (1 BYTE)
017 - MOVSRJ      =  7448 NSEC.  (2 BYTES)
017 - MOVSRJ      = 14983 NSEC.  (5 BYTES)

257 - MAP         =  1334 NSEC.

700 - CONI PI     =  6967 NSEC.
700 - CONO PI     =  1782 NSEC.
700 - DATAI APR   =  6355 NSEC.
700 - DATAO APR   =  6355 NSEC.
```

```
700 - CONO PAG    =  9506 NSEC.
700 - DATAO PAG   = 23032 NSEC.  (LOAD UBR)
700 - DATAO PAG   =  1507 NSEC.  (LOAD AC BLK)
700 - CONI 774    =  7170 NSEC.
700 - CONO 774    =  1787 NSEC.
700 - DATAI 774   =  6594 NSEC.
700 - DATAO 774   =  6388 NSEC.

TEST COMPLETED

D20MON CMD -
D20MON CMD - XXX

D20MON CMD - XXX

D20MON CMD -
```

```
dfkfb

PROGRAM NOT FOUND - DFKFB.

D20MON CMD - r
DISK:DIRECTORY - ps:


D20MON CMD - dfkfb
DFKFB.A10   VER 0.1   02-MAY-75

PDP-10 KL10 INSTRUCTION TIMING TEST (DFKFB)
VERSION 0.1, SV=0.1, CPU#=2123, MCV=275, MCO=40, HO=36, 60HZ

SWITCHES = 000000 000000
CLK SOURCE = EXTERN, CLK RATE = FULL, AC BLK 0 , CACHE: 0 1 2 3

1 - BASIC CLOCK CYCLE IS 33 NSEC.
2 - INDEXING TAKES 34 NSEC.
3 - INDIRECT TAKES 234 NSEC.
4 - INDEXING AND INDIRECT TAKES 267 NSEC.
5 - MOVEI TAKES 266 NSEC.
6 - MOVE FROM AC TAKES 366 NSEC.
7 - MOVE FROM MEMORY TAKES 400 NSEC.
8 - HRR FROM MEMORY TAKES 433 NSEC.
9 - SETOM 0 TAKES 466 NSEC.
10 - JRST TAKES 300 NSEC.
11 - JSR TAKES 567 NSEC.
12 - PUSHJ TAKES 701 NSEC.
13 - ADD FROM MEMORY TAKES 433 NSEC.
14 - MUL (9 ADD/SUB - 18 SHIFTS) TAKES 2.10 USEC.
```

```
15 - DIV TAKES 4.65 USEC.
16 - FIX A FLOATING POINT ONE TAKES 867 NSEC.
17 - FLTR AN INTERGER ONE TAKES 1.53 USEC.
18 - FAD (1 RIGHT SHIFT) TAKES 1.57 USEC.
19 - FAD (8 SHIFT RIGHT - 3 LEFT) TAKES 1.80 USEC.
20 - FMP (7 ADD/SUB - 14 SHIFTS) TAKES 2.33 USEC.
21 - FDV TAKES 4.77 USEC.
22 - DMOVE FROM MEMORY TAKES 733 NSEC.
23 - DFAD (1 RIGHT SHIFT) TAKES 2.03 USEC.
24 - DFAD (8 SHIFT RIGHT - 1 LEFT) TAKES 2.03 USEC.
25 - DFMP (7 ADD/SUB - 32 SHIFTS) TAKES 4.20 USEC.
26 - DFDV TAKES 8.60 USEC.
27 - CONO PI TAKES 1.60 USEC.
28 - CONI PI TAKES 2.80 USEC.
29 - DATAO APR TAKES 1.30 USEC.
30 - DATAI APR TAKES 1.47 USEC.
31 - MOVE TO MEMORY TAKES 566 NSEC.
32 - LOGICAL SHIFT (35 PLACES LEFT) TAKES 533 NSEC.
33 - LOGICAL SHIFT (35 PLACES RIGHT) TAKES 633 NSEC.
34 - LOGICAL SHIFT COMBINED (71 PLACES LEFT) TAKES 933 NSEC.
35 - LOGICAL SHIFT COMBINED (71 PLACES RIGHT) TAKES 967 NSEC.
36 - INCREMENT BYTE POINTER TAKES 834 NSEC.
37 - INCREMENT AND LOAD BYTE TAKES 1.20 USEC.
38 - INCREMENT AND DEPOSIT BYTE TAKES 1.50 USEC.
39 - JFCL TAKES 733 NSEC.
40 - CAI TAKES 400 NSEC.
41 - JUMP TAKES 400 NSEC.
42 - CAM TAKES 500 NSEC.
43 - EQV AC TO AC TAKES 400 NSEC.
44 - EQV MEMORY TO AC TAKES 433 NSEC.
45 - SETOB TAKES 566 NSEC.
46 - AOS TO MEMORY TAKES 700 NSEC.
47 - EXCHANGE AN AC WITH AN AC TAKES 533 NSEC.
48 - EXCHANGE AN AC WITH MEMORY TAKES 700 NSEC.
49 - EXECUTE TAKES 533 NSEC.
50 - BLT MEMORY TO MEMORY TAKES 1.60 USEC.
51 - BLT AC TO MEMORY TAKES 1.57 USEC.
52 - DATAI TAKES 10.00 USEC.
53 - DATAO TAKES 9.00 USEC.

TEST COMPLETED

D20MON CMD -
```