SLAC-PUB-1072
CS 294
(MISC)
July 1972

MICROPROGRAMMED IMPLEMENTATION OF COMPUTER MEASUREMENT TECHNIQUES

Harry J. Saal and Leonard J. Shustek
Stanford Linear Accelerator Center

and

Computer Science Department
Stanford University, Stanford, California 94305

Abstract: Microprogramming has been accepted as a valuable tool in several areas of system design. However, microprogramming has not generally been used as a tool for evaluating the performance of computer systems. This paper describes the implementation of several techniques useful for program monitoring, debugging and system measurement using the microprogramable features of an existing computer system. The measurement system is completely transparent to almost all target programs. Given an existing system with a writable control store, a microprogram measurement system may be the most flexible, inexpensive, reliable, and high-speed means of monitoring the performance of a computer system.

OUTLINE

1. Introduction and Overview

Microprogramming has been accepted as a valuable tool in several areas of system design. It is considered highly valuable for the implementation of emulators for instruction set processors. Where appropriate, it can provide a very cost effective means of implementing a sophisticated instruction repertoire in a highly reliable fashion. As well, microprogramming recently has been looked to as a means of extending a basic manufacturer provided instruction set by the implementation of additional application oriented instructions or processing subroutines. In addition, microprogramming (particularly coupled with writable control store systems) is valuable for research and experimentation in the design of new computer architectures and processor organizations. However, microprogramming has not generally been used as a tool for evaluating the performance of computer systems.

This paper describes the implementation of several techniques useful for program monitoring, debugging and system measurement using the microprogramable features of an existing computer system. The measurement system is completely transparent to almost all target programs. We address ourselves to the designer of computer systems, the user of microcomputer systems and to individuals responsible for the hardware design of microcomputers themselves.

Computer system design has emerged from the realm of intuitive, seat of the pants design. Designers can now take great advantage of highly detailed and reliable measurements based on past and present utilization of existing computer systems. Computer programming can now share many of the same tools and techniques developed to aid the computer designer. Programmers may substantially improve the performance of significant programs given the proper information concerning the execution history of their programs, instructions utilized, interrupt frequencies, etc. These effects are far from minor; one can easily achieve a substantial improvement in a complex program with several hours work and in so doing improve the performance of a software system by several factors.

Work partially supported by the U. S. Atomic Energy Commission.

(To be presented at the 5th Annual Workshop on Microprogramming, University of Illinois at Urbana-Champaign, September 25-26, 1972)

Computer designers face many critical decisions prior to fixing upon a given computer architecture or establishing the best parameters for particular techniques used in implementing a computer design (such as page size or instruction stack depth). We would like to determine the importance and utility of local or relative addressing as opposed to a more general addressing scheme. How much use is made of the various index registers in modern computer architectures? Shall we use a variable or fixed size instruction format? Can we establish which operation codes are most significant in determining the performance of a computer so we may concentrate our design efforts in the area that is most fruitful? We may observe omissions in or improvements for an instruction set by monitoring the serial usage of operation codes in an attempt to observe strong correlations. Designers of memory systems utilizing caches, paging, or overlay schemes need accurate information on the locality and time history of program executions. Operating system designers can take great advantage of information on Input/Output activity, channel and device utilization, etc.

These questions only begin to touch upon the multitude of decisions faced in the design of any modern computer system. We emphasize, however, that these decisions can be based on substantial and accurate measurements and that these measurements, as we will see, can be done reliably and inexpensively using microprogrammed techniques. They should be available to all who choose to take advantage of them without incurring extreme costs or performance degradation.

For the remainder of this paper we will not consider the exact use to which such measurements may be put. Several examples of information available from such measurements will be provided, but they will in no way attempt to suggest the entire spectrum of data analysis that is appropriate for computer design and effective usage.

2. Criteria for Measurement Systems

To be effective, any measurement technique should meet these four criteria:

1. It should be inexpensive and easy to use.

2. It should be easy to modify so that exploratory investigations can be quickly refined in areas of interest.

3. It should enable one to measure any program which could be run on the original non-measured system. In this manner, one can investigate a large number of programs without restricting oneself to those written in one particular language.

4. It should not disturb the original program being measured to any significant degree.

3. Previous Techniques for Computer Measurement

Computer measurement techniques are not a new subject. An excellent survey of traditional techniques may be found in the paper by V.G. Cerf (CERF70).

a) Hardware instrumentation systems

Hardware analyzers (ROTH61, BONN69, COMP70) do not interfere with the system being measured nor do they place constraints on the types of programs which may be monitored. Their major limitations, however, are that they require additional external hardware and substantial knowledge of interfacing techniques in order to extract the appropriate signals and register contents useful for later analysis. Their flexibility is a function of their cost and the extent to which one is willing to attach outboard hardware to an existing computer system. Certainly, they have achieved substantial acceptance in those areas, such as channel or central processor utilization, where appropriate signals can easily be extracted from the lights on the maintenance console of modern computer systems. More sophisticated applications of hardware techniques have been proposed (ESTR67) but have not seen any widespread acceptance.

b) Software instrumentation systems

Software techniques generally fall into two sub-categories, imbedded and external. The imbedded, or in-line measurement technique, has often been applied at the level of the source program language (INGA71, WORT72). By writing a pre-processing system for, say, FORTRAN, one can insert monitoring statements into the original program itself. These counting statements accumulate information concerning the execution history at the source language level. One may also explicitly imbed data collection points in the body of programs, such as an operating system (DENI69, PINK69). This technique, although easy to implement, and quite useful to programmers, language designers, and compiler writers, does not provide appropriate measurement data for the computer system designer.

External software methods generally utilize a sampling technique (JOHN71, STEV68). A regularly scheduled clock interrupt is used to activate a monitor process which samples information such as the program counter, type of instruction being executed, or Input/Output activity in progress for later summarization. Such sampling techniques, however, greatly restrict the type of data that can be collected. Determination of information such as operation code utilization and serial correlations, or branch distances, is done in a statistical fashion and consequently may not be of value for ascertaining appropriate parameters such as cache or page sizes in a computer system.

By far, the most flexible software technique is that of complete interpretation (ALEX72, BUSS70). Interpretive techniques provide the maximum flexibility one can achieve but exhibit substantial costs as an undesirable side effect. Since they increase program execution time by several orders of magnitude, they are often extremely expensive to

use. In addition, not all programs can easily be interpreted. Input/Output activity, program interrupts, etc. can often be the bain of an otherwise excellent interpreter in its attempt to monitor the execution of a large number of computer programs. Because of these difficulties, it has not been economical to conduct measurement studies on a very extensive sample of arbitrary programs using a conventional interpreter.

4. Microprogrammed Measurement Techniques

a) General capabilities and advantages

The overwhelming advantage of microprogram measurement techniques over software interpretation, in our opinion, is the assurance that programs are still interpreted correctly. The kinds of modifications we will describe can be made to an existing emulator with assurance that the semantics of the original emulator are maintained. This is true for all programs regardless of unusual circumstances (such as program interruptions due to underflow) and the presence of Input/Output operations. It is an extremely difficult task to verify that a software interpreter is, in fact, correct for any program which is supplied. On the other hand, an instrumented microprogrammed emulator will even reproduce any errors or unusual interpretations given to instructions that are not properly documented by a computer manufacturer. This assurance is especially valuable in any attempt to measure an extremely wide base of programs with absolutely no modification or restrictions on their behavior.

Microprogram measurement techniques can collect data at an extremely high rate; this promotes substantial use of these techniques and, as well, provides a much closer approximation to real time behavior of an existing system. (In fact, post-measurement analysis of measured data is far more lengthy than the collection of the interpreted data itself). These techniques do not require the existence of any operating system for performing data collection and device handling. Thus we can measure programs which contain bugs or are operating systems themselves with no restriction. Conventional interpreters generally cannot be used to interpret an entire computer system at once.

Microprogram techniques may be used to monitor any one of the programs in a multiprogrammed system through slight modifications of the system environment. We recommend, for example, that one bit in a program status word be dedicated to enabling and disabling the program monitor. The microprogram system can then selectively monitor one of a large number of programs which is being controlled by a complex operating system.

b) Examples

We have constructed a series of measurement microprograms for the Standard Computer Corporation IC7000 computer system (STAN69)*. These were intended to illustrate some of the possible techniques; others are undoubtedly just as feasible and useful.

i. Type of data gathered

Two classes of instrumentation systems were implemented on the IC7000 to collect data. The first of these produced a tape from which we extracted the complete history of the execution of programs by recording all successful branch instructions and relocation information. The second accumulated, among other things, the distribution of individual operation code usage and a matrix of operation code pair executions.

ii. Sample analysis of measurement data

Some brief samples of straightforward analyses of typical measurement data are shown in Figures 1 through 3. Figure 1 indicates the distribution of use of main storage by a given software module. Correlating this information with the user's program quickly leads him to focus on those areas which deserve maximum attention and optimization. The branch trace data can also be used to provide an unusual debugging aid to the user since after the completion of execution trace measurement, it is possible to print a list of the last one hundred branches executed by a given program. This has proved extremely valuable for understanding the sequence of events leading to a program interrupt which then caused termination of a user's program. We have also incorporated as the standard microprogram for our IC7000 a compatible emulator which always maintains (in control store) the addresses of the last two branches successfully taken by each of the Central Processor and the Input/Output Processor. It would be straightforward to extend the instruction set to allow access to this pair of words for use by a target debugging system.

Figure 2 shows a plot of operation code utilization on a dynamic basis for a variety of measured programs in the CPU. The operation codes were sorted by frequency of use separately for each program measured. The vertical axis indicates the fraction of instructions not accounted for by the number of sorted operation codes shown on the horizontal axis. This data may be compared to that collected on the CDC3600 by Foster et al (FOST71). Figure 3 indicates the frequency of pairs of operation codes executed during a FORTRAN compilation. We have shown only those transitions which were greater than 1 percent of the total number of instructions executed. This simple structure represents over 80 percent of all instruction pairs executed by the compiler. This type of information in extremely useful in deciding which microprogram routines are the critical ones in determining processor speed, and which are candidates for control store swapping or replacement by target-level programs.

We reserve comments and analysis of this data for future publication and merely present some examples of the most straightforward type of analysis possible from our microprogrammed measurements.

*This system has been provided for our research purposes by the Standard Computer Corporation of Santa Ana, Calif.

Microprogramming provides a convenient alternative to software interpretive techniques. A microprogrammed emulator is, indeed, an interpreter for a computer instruction set. If we can modify an existing emulator in such a manner as to collect our desired measurement information without introducing any changes to the semantics of the emulation itself, then, (except for possible time dependent problems) by definition, we can correctly execute any program which would run on the original emulator.

iii. Other possible measurements

There are a variety of other aspects of computer utilization which can also be effectively measured using microprogrammed techniques. Many of these, when used without simultaneously measuring high rate events such as operation code tracing, have a negligible effect on system performance and may therefore be continuously used to monitor activity and make information available to the software. Channel and device statistics are particularly easy to accumulate; channel wait and overlap time, which must usually be gathered by an external hardware monitor, is a typical example.

Another area which can profit from the use of such microprogrammed techniques is software debugging. Many of the devices which are available only at the console or from a software interpreter, such as address and data reference stops, can be implemented in the microprogram and controlled by the software. Breakpoints can be established without modifying main storage, thus allowing even self-modifying programs to be debugged with these tools.

5. Implementation on the Standard Computer Corporation IC7000

Figure 4 indicates the organization of the IC7000 system. It is comprised of two independent processing units each of which contains a writable control store of 2048 18-bit words of vertical microinstructions. These two processors have entirely different microcomputer organizations, as well as different responsibilities for program execution and Input/Output processing but share a 64K 36-bit main memory. Rather then describe the system in greater detail at this point, we will mention those features which are relevant at the appropriate time. The reader can refer to STAN69 for further information. The microprograms we have written (SHUS72) provide an environment almost identical to the original one upon which our system software executes. The only differences are in (a) speed of execution, (b) 2000 words of main storage buffer space now unavailable to user programs, and (c) one tape drive which cannot be used by a target level program. Using these measurement microprograms is utter simplicity; one simply loads the writable control store from a different microprogram tape, ensures that a scratch tape is available for the measurement data, and then, at the appropriate point, activates data collection by depressing a maintenance switch not otherwise used by any system software.

a) Data collected by measurement microprograms

Currently measurement can be performed using one of four instrumented microprograms. For either the Central Processing Unit (CPU) or the Input/Output Processor (IOP), data may be collected on either program instruction execution or operation code utilization.*

The execution trace microprogram is generally limited by the speed of the output tape unit. Consequently, we have attempted to compress the measured data at the microprogram level rather than accumulating unprocessed raw data. For example, simple loop structure is detected by recording the address of the last branch instruction and its target instruction. An output word is generated whenever either of these two quantities are changed. Multiple branches in loop mode are encoded with the appropriate count prior to being entered in the output buffer.

The Central Processing Unit contains hardware program and data address relocation registers. Typically we are interested in summarizing data for a given user program regardless of where it has been moved about in main storage. Therefore it is necessary to include in the measurement data information concerning the relocation status of the Central Processor. All pertinent information is recorded whenever changes in relocation are made. The analysis routine can then follow a program as it is moved about in the physical address space of core by the time-sharing system. All addresses recorded are virtual addresses but they can be converted to physical addresses at analysis time.

Two types of operation code summaries are accumulated during the execution of a program. The first is a frequency count of the execution of individual operation codes. For the Input/Output Processor, this is a straightforward task since the entire operation code information is contained in the most significant eight bits of an instruction. The number of words of main storage allocated to operation code frequency counts is therefore quite acceptable.

This is not the case with the Central Processing Unit, which closely resembles an IBM 7090. There are basically two types of operation codes, the short and long formats, distinguished by the first three bits. Six of the eight major prefix operation codes are considered individual instructions. The remaining two operation codes utilize up to an additional nine bits for further decoding. We could not afford to accumulate frequency data for the full 12 bit operation code field due to the limited size of main storage. Consequently, the microprogrammed measurement routine decoded and separated the short and long form operation codes, thus using 6 plus 2 x 2**9 entries in the frequency histograms. This type of flexible encoding demonstrates one of the significant advantages of a microprogram measurement technique when compared to hardware

*Additional information is also collected whenever an Input/Output operation is issued. We will not discuss these measurements in any detail in this paper.

monitoring. Based on early measurement of operation code utilization, a 32 x 32 matrix of operation code pairs was also accumulated. Over 90 percent of the instruction pair sequences could be uniquely measured using this size matrix.

b) Use of special hardware features

Many unusual techniques had to be developed in order to efficiently capture the kinds of measurement data to which we have previously referred. We were able to take advantage of some special hardware features which were present in our microcomputer system. One might think it an easy matter to insert measuring routines at all appropriate places in an emulator. In principle, this is the case, but since microprogram storage is an extremely scarce commodity, it was prohibitively expensive to insert measurement routines throughout the microprogram. Since our microcomputers possess a limited subroutining facility at the microprogram level, it was not even feasible to include a subroutine call at every point at which we wished to measure the performance of the system. In addition, many instructions are executed directly in hardware at instruction fetch time (most of the program transfer instructions). Others share common microcode but are semantically distinguished by a large number of flip-flops (set by the hardwired instruction fetch and decode) which perform extensive residual control.

In the case of the Input/Output Processor, we were able to take advantage of a special hardware trace facility. When enabled by a switch in the maintenance console, the IOP microprocessor automatically traps to a special control storage location, immediately prior to the execution of a target instruction. The measurement routine at that location can read the current program counter and the instruction to be executed next. After posting the appropriate data in the main storage buffer, the trace microroutine then exits to the hardwired scheduling sequence. This time, however, the instruction is fetched and decoded by the hardware and executed normally by the conventional microprogram. When the emulation is complete, the measurement routine again receives control.

The Central Processing Unit trace program was able to take advantage of a completely different mechanism present in the CPU. Under microprogram control we enabled a mode wherein every transfer instruction caused a microprogram trap to occur rather than continuing normal execution. We could then easily accumulate execution trace data by inserting one microroutine for handling all transfer instructions. It was necessary as well to accumulate all relocation changes as has been described previously; this required little modification to the original emulator.

Operation code tracing was substantially more difficult in the CPU than the IOP. There was no hardware facility available which permitted us to gain control in a centralized location prior to each instruction execution. We were, however, able to simulate such behavior using completely different hardware techniques.

In order to directly emulate the delaying of an interruption in the central processor following the re-enabling of interrupt or loading relocation status, the following hardware controls were available:

1. It was possible to inhibit a micro-level interrupt for one target level instruction execution.

2. It was also possible to force a special microprogram interrupt to take place after completion of a microroutine.

Through the combination of both of these features, a trace routine is able to gain control, request that an interrupt take place upon exit from the trace routine, but to delay this interrupt until after the following instruction execution. In this manner, we are able to alternately trace and execute instructions fetched from main storage. This is a substantial advantage compared to the possibility of having to include measurement microcode in every emulation routine or to not utilize the hardware instruction decoding which was available and used by our emulators. As was mentioned earlier, even after the operation code information was available considerable encoding was required in order to minimize the amount of main storage used for accumulating frequency histograms.

The microprogrammer is in a continual battle to overcome the limitations of available control store. This lack of freedom to incorporate new measuring routines may rule out the possibility of extensive microprogram measurement techniques on some computer systems. We have tried to point out some special hardware techniques which substantially reduce the requirement for additional control storage. Put another way, we recommend to the designers of microcomputers that they incorporate the kinds of hardware tracing facilities mentioned above in order to facilitate the implementation of measurement techniques. Providing such hardware is not a major additional expense in the design of a microcomputer system, yet it provides an extremely powerful tool for the system designer.

c) Difficulties encountered and recommendations for their solution

Dealing with the Input/Output conflict between the microprogram measuring routine and the system being measured was the single most difficult problem in the implementation. The microprogram has a formidable task in attempting to appear completely transparent to an emulated program which utilizes the same IO channels as the trace program. This stems from the nature of the interrupt and status registers as implemented in the Input/Output Processor. The emulated program expects to have access to a variety of channel and controller status registers appropriate to any Input/Output operation it has initiated. Consequently, it is necessary for the measurement microprogram to save these registers prior to any operations which it initiates on its own behalf. Whenever a target level program requests the contents of such registers, we must provide the values saved prior to our own interference. The microprogram must dinstinguish interrupts caused by target level IO operations from those resulting from operations initiated by the measurement routines. It also

must defer initiation of channel operations whenever concurrent activity is attempted by both the target level and measurement routines.

We would recommend a less ambitious approach whenever possible. If the measurement data can be collected on an Input/Output channel which is not used by the programs being measured, there can be a substantial reduction in the complexity of modifications necessary to an existing microprogrammed emulator. Properly dealing with the interference of target Input/Output simultaneously occurring on the channel utilized for measurement output is a major problem facing the microprogram measurement implementer. The exact complexity of this problem is both a function of microprocessor hardware features and the generality of the microprogram measurement tools. We can provide little advice other than indicating that this is an extremely difficult problem to master correctly.

Microprogram machines are generally not completely microprogrammed. Many aspects of instruction decoding and operand fetching may be performed in a hardwired scheduler in the interest of increased efficiency. This technique conflicts with microprogram measurement. The hardwired decoding scheme may automatically set a variety of residual control registers and flip-flops to simplify the semantic emulation routines. Current microprocessors have not been designed to allow these registers to be explicitly read by an emulator and thus they are not available to measurement routines. This lack of generality imposes unnecessary complications to the microprogrammer, but could be avoided in future microprocessor design.

The conflicts of main storage utilization by the measurement routines and user programs can be solved in a number of ways. Our system contains an instruction which returns the size of available core storage in any configuration. By providing such an instruction and ensuring that all systems utilize it in their initialization, the conflict over main storage buffers may easily be resolved.

A severe problem found in the implementation of extensions via microprogramming, generally not found in conventional software interpreters, arises from the lack of many general facilities at the microprogram level. Microprocessers may have no, or at most limited, subroutine calling mechanism. There may not be any spare internal registers which can be used for local computation by the microprogram measurement routines without saving them. These problems are not insuperable but they seriously impact both the time and space requirements of the measurement routines. In addition, microprogrammers do not have an elaborate data manipulation instruction set available for programming, and it is extremely difficult to utilize the existing target level instruction emulation routines. Data compression at the microprogram level often contributes substantially to the degradation of processor performance during measurement by microroutines. Conventional interpreters generally do not contain large incremental degradation at this phase of processing since their overall performance is so poor.

d) Timing Estimates

Instruction tracing is limited by the speed of our output tapes. We degrade performance by a factor of 20 using a 60 KB 7-track tape drive for recording trace output. With the tape drive disabled, the system runs roughly two times slower than normal execution speed. The results of our analysis have indicated that approximately 50 percent of all branches change the program counter by less than eight locations. Consequently, we could get substantially greater density of information on the tape if we were to use more sophisticated encoding techniques in addition to simple loop detection. Operation code measurements are not tape bound, since we accumulate frequency distributions in core during the execution of a program. This data collection slows down Central Processor execution by somewhat over a factor of 10 to 1 due to the difficulty of compacting the operation codes.

6. Summary

Measurements using microprogramming techniques should not be viewed as a replacement for existing methods. They do, however, offer substantial advantages in meeting the four criteria originally presented for an effective measurement system. Given an existing system with a writable control store, a microprogram measurement system may be the most flexible, inexpensive, reliable, and high-speed means of monitoring the performance of a computer system. It is possible to incorporate, under user control, microprogrammed measurement techniques within the framework of existing operating systems.

We recommend the development of microprogram versions of proposed computer organizations so that detailed evaluation can be performed by microprogram techniques, as illustrated above. The data available through these techniques can be used to make sound decisions regarding the appropriate tradeoffs between hardwired, microprogrammed and software implementations of the architecture of future computer systems.

REFERENCES

ALEX72 Alexander, W. G., "How a Programming Language Is Used", University of Toronto, Computer Systems Research Group Technical Report CSRG-10, February 1972

BONN69 Bonner, A. J., "Using system monitor output to improve performance", IBM System J., Vol. 8, No. 4, 1969, pp. 290-298.

BUSS70 Bussell, B., and Koster, R. A., "Instrumenting Computer Systems and Their Programs", AFIPS FJCC Proceedings, 1970, pp. 525-534

CERF70 Cerf, V. G., 'Measurement of Recursive Programs", UCLA School of Engineering and Applied Sciences Report No. 70-43, May 1970

COMP70 Computer Synectics, Inc., Santa Clara, Calif., "System Utilization Monitor User's Manual", M-1001, Nov. 1970.

DENI69 Deniston, W. R., "SIPE: A TSS/360 Software Measurement Technique", Proc. ACM 24th Nat'l. Conf., 1969

ESTR67 Estrin, G., Hopkins, D., Coggan, B., and Crocker, S., "Snuper Computer - A Computer Instrumentation Automaton", AFIPS SJCC Proceedings, 1967, pp. 645-656

FOST71 Foster, C. C., Gonter, R. H., and Riseman, E. M., "Measures of Op-Code Utilization", IEEE Transaction on Computers, May 1971, pp. 582-584

INGA71 Ingalls, D. H., "FETE: A Fortran Execution Time Estimator", Stanford University Computer Science Department Report STAN-CS-71-204, February 1971

JOHN71 Johnson, R., and Johnston, T., "PROGLOOK Users Guide", Stanford University Computation Center, Document No. SCC-007, October 1971

PINK69 Pinkerton, T. R., "Performance Modeling in a Time-Shared System", CACM Vol. 12, No. 11, pp. 608-610, November 1969

ROTH61 Roth, B., "Channel Analysis for the IBM 7090", Proceedings ACM 16th Nat'l. Conf., 1961

SHUS72 Shustek, L., "Measurement Miniflow", Stanford Linear Accelerator Center Computation Group, Technical Memo CGTM-132, February 1972

STEV68 Stevens, D. F., "System Evaluation on the Control Data 6600", Proc. IFIP Congress, Software Session II, Booklet C, 1968, pp. 34-38

STAN69 Standard Computer Corporation, Santa Ana, Calif., "IC7000 System Summary", Form 807010-3; "IC7000 SPU Inner Computer - Principles of Operation", Form 807003-2; "IC7000 ALP Inner Computer - Principles of Operation", Form 801003-4

WORT72 Wortman, D., "A Study of Language-Directed Machine Design", PhD Thesis, Stanford University, 1972

Fig. 1--Display of executed locations

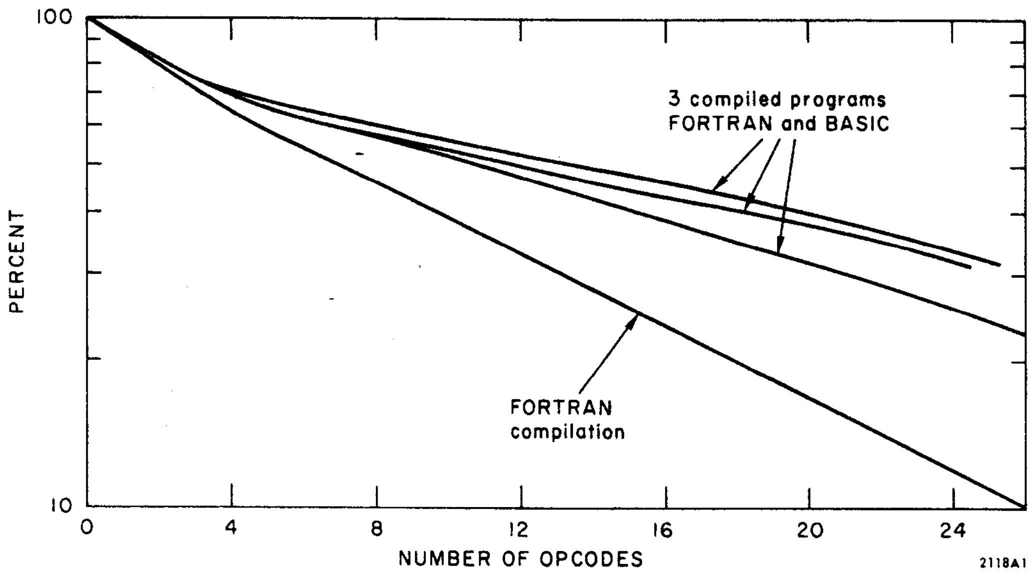

Fig. 2-- Log-survivor function of instructions executed vs. sorted operation codes of IC7000 CPU.

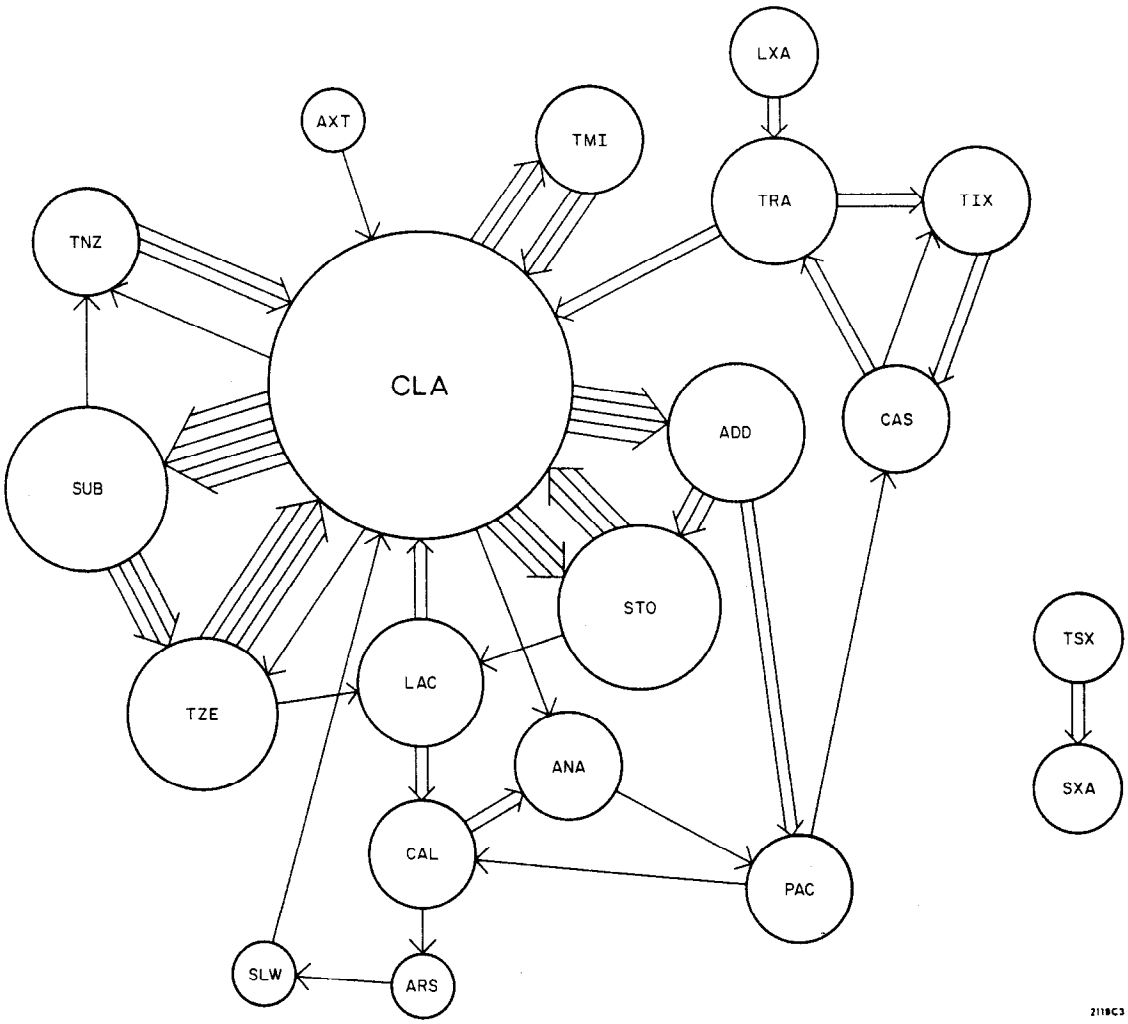

Fig. 3--IC7000 operation code pairs with frequency > 1%.
[The number of lines connecting two operation codes is approximately the transition frequency in percent; the area of each operation code is proportional to its total frequency.]

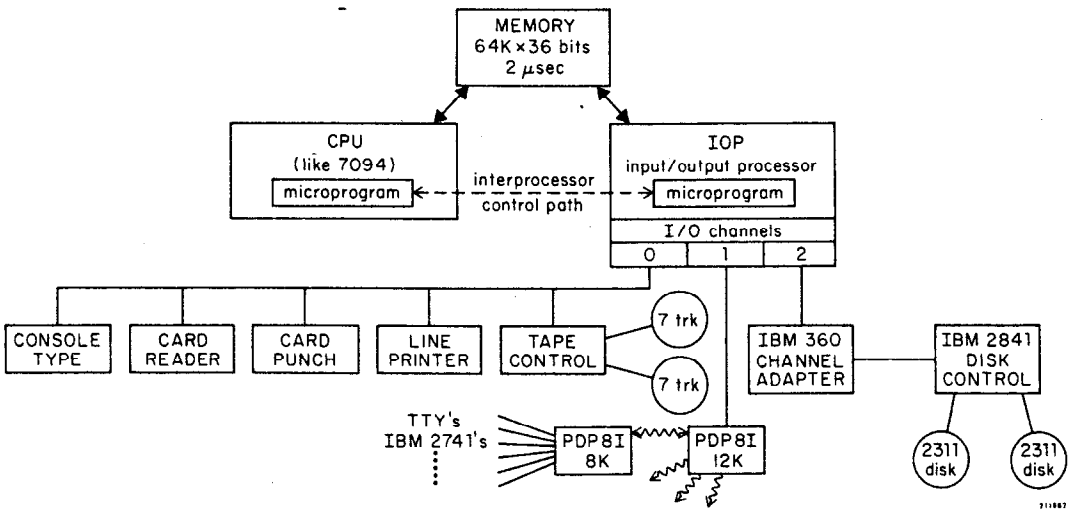

Fig. 4--IC7000 configuration