MEMORANDUM TO P. M. MORSE PROPOSING TIME SHARING

John McCarthy, Stanford University

January 1, l959

To: Professor P.M. Morse
From: John McCarthy
Subject: A Time Sharing Operator Program for our Projected IBM 709

1 INTRODUCTION

This memorandum is based on the assumption that MIT will be given a transistorized IBM 709 about July l960. I want to propose an operating system for it that will substantially reduce the time required to get a problem solved on the machine. Any guess as to how much of a reduction would be achieved is just a guess, but a factor of five seems conservative. A smaller factor of improvement in the amount of machine time used would also be achieved.

The proposal requires a complete revision in the way the machine is used, will require a long period of preparation, the development of some new equipment, and a great deal of cooperation and even collaboration from IBM. Therefore, if the proposal is to be con- sidered seriously, it should be considered immediately. I think the proposal points to the way all computers will be operated in the future, and we have a chance to pioneer a big step forward in the way computers are used. The ideas expressed in the following sections are not especially new, but they have formerly been con- sidered impractical with the computers previously available. They are not easy for computer designers to develop independently since they involve programming system design much more than machine design.

2 A QUICK SERVICE COMPUTER

Computers were originally developed with the idea that programs would be written to solve general classes of problems and that after an initial period most of the computer time would be spent in running these standard programs with new sets of data. This view completely underestimated the variety of uses to which computers would be put. The actual situation is much closer to the opposite extreme, wherein each user of the machine has to write his own program and that once this program is debugged, one run solves the problem. This means that the time required to solve the problem consists mainly of time required to debug the program. This time is substantially reduced by the use of better programming languages such as Fortran, LISP (the language the Artificial Intelligence Group is developing for symbolic manipulations) and COMIT (Yngve's language). However, a further large reduction can be achieved by reducing the response time of the computation center.

The response time of the MIT Computation Center to a performance request presently varies from 3 hours to 36 hours depending on the state of the machine, the efficiency of the operator, and the backlog of work. We propose by time sharing, to reduce this response time to the order of 1 second for certain purposes. Let us first consider how the proposed system looks to the user before we consider how it is to be achieved.

Suppose the average program to be debugged consists of 500 instructions plus standard subroutines and that the time required under the present system for an average debugging run is 3 minutes. This is time enough to execute 7,000,000 704 instructions or to execute each instruction in the program 14,000 times.

Most of the errors in programs could be found by single-stepping or multiple-stepping the program as used to be done. If the program is debugged in this way, the program will usually execute each instruction not more than 10 times, 1/1400 as many executions as at present. Of course, because of slow human re- actions the old system was even more wasteful of computer time than the present one. Where, however, does all the computer time go?

At present most of the computer time is spent in conversion (SAP-binary, decimal-binary, binary-decimal, binary-octal) and in writing tape and reading tape and cards.

Why is so much time spent in conversion and input output.

1. Every trial run requires a fresh set of conversions.

2. Because of the slow response time of the system it is necessary to take large dumps for fear of not being able to find the error. The large dumps are mainly unread, but nevertheless, they are necessary. To see why this is so, consider the behavior of a programmer reading his dump. He looks at where the program stopped. Then he looks at the registers containing the partial results so far computed. This suggests looking at a certain point in the program. The programmer may find his mistake after looking at not more than 20 reg-

> isters out of say 1000 dumped, but to have predicted which 20 would have been impossible in advance and to have reduced the 1000 substantially would have required cleverness as subject to error as his program. The programmer could have taken a run to get the first register looked at, then another run for the second, etc., but this would have required 60 hours at least of elapsed time to find the bug according to our assumptions and a large amount of computer time for repeated loading and re-runnings. The response time of the sheet paper containing the dump for any register is only a few seconds which is OK except that one dump does not usually contain information enough to get the entire program correct.

Suppose that the programmer has a keyboard at the computer and is equipped with a substantial improvement on the TXO interro- gation and intervention program (UT3). (The improvements are in the direction of expressing input and output in a good programming language.) Then he can try his program, interrogate individual pieces of data or program to find an error, make a change in the source language and try again.

If he can write program in source language directly into the computer and have it checked as he writes it, he can save additional time. The ability to check out a program immediately after writing it saves still more time by using the fresh memory of the programmer. I think a factor of 5 can be gained in the speed of getting pro- grams written and working over present practice if the above- mentioned facilities are provided. There is another way of using these facilities which was discussed by S. Ulam a couple of years ago. This is to use the computer for trial and error procedures where the error correction is performed by a human adjusting parameter.

The only way quick response can be provided at a bearable cost is by time-sharing. That is, the computer must attend to other customers while one customer is reacting to some output.

3 THE PROBLEM OF A TIME-SHARING OPERATOR SYSTEM

I have not seen any comprehensive written treatment of the time-sharing problem and have not discussed the problem with anyone who had a complete idea of the problem. This treatment is certainly incomplete and is somewhat off the cuff. The equipment required for time-sharing is the following:

a. Interrogation and display devices (flexowriters are possible but there may be better and cheaper).

b. An interrupt feature on the computer—we'll have it.

c. An exchange to mediate between the computer and the external devices. This is the most substantial engineering problem, but IBM may have solved it.

In general the equipment required for time-sharing is well understood, is being developed for various advanced computers, e.g., Stretch TX2, Metrovich 1010, Edsac 3. I would not be surprised if almost all of it is available with the transistorized 709. However, the time-sharing has been worked out mainly in connection with real-time devices. The programs sharing the computer during any run are assumed to occupy prescribed areas of storage, to be debugged already, and to have been written together as a system. We shall have to deal with a continuously changing population of programs, most of which are erroneous.

The major problems connected with time-sharing during pro- gram development seem to be as follows:

1. Allocating memory automatically between the programs. This requires that programs be assembled in a relocatable form and have a preface that enables the operator program to organize the program, its data, and its use of common subroutines.

2. Recovery from stops and loops. The best solutions to these problems require

   Changing the stop instructions to trap instructions. This is a minor modification to the machine. (At least it will be minor for the 704.)

   Providing a real time alarm clock as an external device.

3. Preventing a bad program from destroying other programs. This could be solved fairly readily with a memory range trap which might not be a feasible modification. Without it, there are pro- gramming solutions which are less satisfactory but should be good enough. These include:

   Translations can be written so that the programs they produce cannot get outside their assigned storage areas. A very minor modification would do this to Fortran.

   Checksums can be used for machine language programs.

   Programming techniques can be encouraged which make destruction of other programs unlikely.

   There is an excessive tendency to worry about this point. The risk can be brought down to the present risk of having a program ruined by operator or machine error.

4 SUMMARY

1. We may be able to make a major advance in the art of using a computer by adopting a time-sharing operator program for our hoped-for 709.
2. Such a system will require a lot of advance preparation starting right away.
3. Experiments with using the flexo connection to the real-time package on the 704 will help but we cannot wait for the results if we want a time-sharing operator program in July 1960.
4. The cooperation of IBM is very important but it should be to their advantage to develop this new way of using a computer.
5. I think other people at MIT than the Computation Center staff can be interested in the systems and other engineering problems involved.

/@steam.stanford.edu:/u/ftp/pub/jmc/timesharing-memo.tex: begun 1994 Dec 29, latexed 1996 Sep 9 at 4:15 p.m.