MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Department of Electrical Engineering

An Experiment in the use of the Compatible Time-Sharing System in the Laboratory of a Computer Programming Subject.

Final Report to
The M.I.T. Computation Center
on problem M3479

June 15, 1964
J. H. Saltzer
Final Report: Problem 706479

An Experiment in the Use of the Compatible Time-Sharing System in the Laboratory of a Computer Programming Subject.

I. Summary: This report describes an experiment in which a group of 6.41 students solved a class laboratory problem with the aid of time-sharing consoles. This laboratory problem, a straightforward bit manipulating program, was the first encounter with the FAP language for the class. To aid the students, an automatic grading system was devised, as was a specialised testing program which connected with the FAPDEG command. The students were limited to two hours of console time, and most were able to complete the problem in that time. The average amount of IBM 7094 time used was about 5 minutes. For comparison, a similar group of students solved the same problem using standard batch-processing techniques. This second group used about 1 minute of 7094 time per student. There is no question that under the conditions of the experiment the amount of 7094 time used appears to be prohibitively high, however, there are clear-cut avenues of improvement. A thoughtful analysis indicates that the computer time used for time-sharing class programs need not be too great, comparatively, and there may be dividends in the form of increased understanding with less effort on the part of both the students and instructor.

II. Introduction: In order to obtain scaling information needed to allow intelligent discussion of whether or not time-sharing consoles should be used for computer programming subjects, an experiment was undertaken to learn what techniques might be useful, and how much console and computer time would be needed. The 6.41 Spring term, 1964 class was divided at random into two groups of about 40 students each. One group did the fifth machine problem in the "standard" fashion used for several semesters in 6.41, submitting decks of cards to be run under a batch-processing monitor extended to include a specialised automatic grading program. Each student in this group had four "tries" in which to get his program working. Students in the second group wrote the same program, but debugged it at time-sharing consoles during two successive weekends. Each student in this group had up to four half-hour periods in which to type in his program and debug it.


The students were to write a subroutine in the FAP language which examines the 36-bit word in the MQ register, and reports the length of the longest unbroken string of one-bits. This program, intended as an introduction to the FAP language, can be written in about a dozen instructions, yet is moderately intricate in terms of logical tests which must be performed. It is similar in difficulty to introductory FAP problems given to 6.41 classes in the past.

A number of special programs were written ahead of time to aid the students in debugging their programs. Each student obtained a copy of two such programs, a testing program and a grading program. The students tested their subprograms themselves; when they were satisfied that the subprogram worked, they submitted it to their instructor for a grade with the aid of the grading program. They were permitted only one chance to have their subprogram graded.

III. Support Programs: The student typed in and corrected his program with the standard system INPUT, EDIT, and FILE commands, and translated it with the FAP command. At this point he called into play a special program provided by his instructor named TEST. This program contained two important modules, FAPREG, and a special testing routine (named EVAL) written for the student's subroutine. Also used was a general testing package similar to that used in batch processing operations, which provided clock, output and service routines for the EVAL.

The technique used by the TEST program is as follows:

THE student types the command.

RESUME TEST α

where α is the name of his BSS subroutine. The TEST program moves a BSS into common file one, where a testing library is stored, and chains to LOADGO loading the student's program, a main program, and the testing library. Common file one is used in this application to keep down the size of the TEST program. If TEST contained the testing library, some 20-30 tracks would be necessary to store it; however, as a chaining routine it only takes one track, a considerable saving since each student must have a copy of the TEST program. Another advantage of this technique is that a bug in the testing library can be fixed without having to tell each student to obtain a new copy of the TEST program.


6. The program described in this section were developed under the problem application N3329; listings of the programs will accompany the final report of that problem.

When LOADCO has loaded the student's program and the testing library, it transfers to the main program, which initializes trap returns, etc., and transfers to the specialized testing program for this student subroutine. At this point, the exact procedure used to test the student's subroutine is up to the testing program, EVAL.

In the EVAL used, an initial FAPDBG request file was created, containing requests to work the student's program, and to define the locations of 10 entry points into the EVAL, which were to be used as entries to 10 test cases. The EVAL then chained to FAPDBG.

The student then initiated action by typing FAPDBG requests. The ten symbols "T1" through "T10" corresponded to the 10 test cases which the EVAL was prepared to use on his subroutine. The student, to call for one of these test cases, typed a "go" request:

```
G T1
```

for example. This request returned control to the EVAL which proceeded with test case one. A number was placed in the M2, and control passed to the student's program. In most cases, control returned, the EVAL printed an appropriate comment, and chained to FAPDBG to await further requests. If the student was satisfied with the results of the first test case, he could go on to another. If not, he had the full facilities of FAPDBG to explore his program. (The student was not told of all the facilities of FAPDBG, and consequently used FAPDBG primarily as a post-mortem program.) When he uncovered an error, or was finished with all ten tests, he typed a "Q" request to exit from FAPDBG and went on to make corrections, logout, or have his program graded.

When a student was satisfied with his program, he submitted it to his instructor by typing

```
RESUME GRADE a BSS
```

The saved file GRADE moved his file, "a BSS" into common file two, then went into a SLEEP loop, every thirty seconds awakening to look in common file three for a file "a BM". The instructor was logged in at another console, running a program named LOOK. This program was also in a SLEEP loop, looking in common file two for programs to grade. When a program was found, it was graded by running it with a test program similar to the one the student used for testing, but with different test cases. (Since a random number generator cannot be used to select test cases at random the particular set of test cases used depended on the date, time, and length of the student's program.) The results of this grading run were placed in a file "a BM" in
common file three, and the student's grade typed out on the
instructor's console. The LOOK program then returned to
its SLEEP loop. When the student's GRADE program discovered
the "a file" in common file three, it printed its contents
on the student's console, and logged him out, as he had
no further use for the facilities.

The technique used here has several virtues. First,
it is completely automatic. If desired, the instructor can
be left logged in and LOOK'ing for a period of time, without
direct supervision and without using much computer time.
It also affords a degree of security, since the student
cannot interact with his program while it is being graded.
Also, it is very difficult for him to affect his grade,
except by writing a correct program. (With the addition of
the memory protect feature, it should become impossible for
him to breach any security boundaries.)

One difficulty remains in the usage of common files
for transmitting the program to and from the instructor:
Another student could interrupt the program (or, less
importantly, the post-mortem) while it is in the common file.
This is only a minor problem, however, in a reasonably
small and friendly class, and will probably be remedied in
future versions of the time-sharing system which have better
bulk message handling techniques.

A significant virtue of both the grading and testing
programs is that the student need know nothing about how
they operate or, for example, about common files. In the
course of using the time-sharing system, he only learns a
vocabulary of about ten commands, and five PAGING requests.

IV. Logistics of the Experiment: The 6.41 class was randomly
split into two groups of about forty students each. One
group was to use time-sharing, the other batch processing.
Each student was instructed to write his own solution to
the problem. Students in the batch processing group punched
their solution on cards and submitted up to four runs at
3-to-4 day intervals over a two week period. Those in the
time-sharing group signed up for half-hour console sessions
on two consecutive weekends. Members of this group were
allowed up to two hours total console time.

Instructors in the course provided the Computation Center
with cards punched with student names, programmer numbers,
passwords, track allotments, etc., to simplify the task of
inserting them into the system. To insure that the primary
emphasis was on console time used, a generous computer time
allotment of 12 minutes was given each student. The students
and their instructor were assigned to a party-line group of
three lines. The students were assigned to a console use group restricting them to teletypes in room 26-265. The restriction to teletypes only resulted from a temporary lack of "here is" codes on the 1050 consoles.

During the weekends of the experiment, an instructor was on duty at all times to answer questions and handle emergencies. As a further aid to the student, a brief memo was written describing basic features of the time-sharing system, ten commands, and five FAPDBG requests. A copy of this memo is included as appendix A of this report.

V. Results: A total of 40 students were in the group which used time-sharing. Four of these students dropped the course before the experiment had finished, and six more did not attempt to do the problem. Thirty students, then, participated in the experiment. On the other side of the ledger, 43 students used to do batch processing. Of these two dropped, and two failed to work on the problem, leaving 39 students participating. The following statistics were collected on the time-sharing group:

<table>
<thead>
<tr>
<th></th>
<th>average of group</th>
<th>spread (standard deviation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Console time used</td>
<td>162 min.</td>
<td>45 min.</td>
</tr>
<tr>
<td>Number of distinct console sessions</td>
<td>2.7</td>
<td>1.4</td>
</tr>
<tr>
<td>Average time per console session</td>
<td>35 min.</td>
<td>not computed</td>
</tr>
<tr>
<td>7094 time used (times typed at logout)</td>
<td>5.6 min.</td>
<td>3.2</td>
</tr>
<tr>
<td>Time spent in INPUT, EDIT, and FILE commands</td>
<td>38 sec.</td>
<td>42 sec (!)</td>
</tr>
<tr>
<td>Number of uses of INPUT-FILE or EDIT-FILE sequence</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>Time spent in FAP command</td>
<td>29 sec.</td>
<td>23</td>
</tr>
<tr>
<td>Number of uses of FAP command</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>Time spent in TEST program (includes loading)</td>
<td>169 sec.</td>
<td>112</td>
</tr>
<tr>
<td>Number of uses of TEST program</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>Time spent in other miscellaneous commands</td>
<td>34 sec.</td>
<td>54 sec.</td>
</tr>
<tr>
<td>Total 7094 time accounted for in above command breakdowns</td>
<td>4.5 min.</td>
<td></td>
</tr>
</tbody>
</table>
These statistics were collected by having the students go over their output listings carefully to note times typed at "READY" by the system. The total 7094 time used by each student was checked by comparison with records kept by the time-sharing system, and in most cases was within about 10 percent of the student's computation of 7094 time used, indicating a reasonable degree of reliability of the student-collected data.

By the way of contrast, the 41 students using the batch-processing technique used a total of 42 minutes of 7094 time for an average of a little over one minute of 7094 time per person. They used an average of 3 runs to complete the problem.

VI. Analysis and Discussion of Results: The statistics of the previous section are to a major extent meaningless until some interpretation is applied to them. Some qualifications are provided here.

1. Computer time used. A bold comparison of computer time used by the two groups indicates that time-sharing is unprofitable by a five-to-one margin. However, we should make several qualifications to these figures.

The most obvious qualification is that of comparing "non-equivalent" jobs. For example, the time-sharing group used typing and editing facilities connected with the computer, and were charged for computer time for these operations. The background group used keypunchers and listing machines for this function; it is clear that in this area the jobs performed by the computer are not equivalent. In a more important sense a different type of education was also accomplished. While both groups had about 80 percent success in getting their programs to work, the group using keypunches worked under the usual handicap of excessive attention to detail; those using time-sharing were able to devote less attention to detail and more to understanding the nature of their problem and learning the FAP language. Students using time-sharing were able to concentrate all their energies of an afternoon or two to the solution of the problem; with batch-processing, a run turned around every three or four days required dropping and picking up the problem several times. Although statistics on hours were not collected, probably fewer student hours were spent accomplishing an equivalent level of learning.

A second qualification is that this was the students' first use of the time-sharing system, and there was certainly some computer time wasted as they became familiar with consoles, commands, editing programs, etc. The
batch-processed group had submitted four earlier machine problems in other languages using batch-processing, and were familiar with the techniques and level of precision required. It seems likely that a second machine problem would have required less time, on the average, for the time-sharing group.

The last qualification is that the figures being compared are for a highly-tuned batch-processing system specially modified for classes, and a "first effort" at time-sharing techniques. The batch-processing system used a special short library and a compressed system tape containing only essential system records. These student jobs were assembled, tested, and given dumps in an average of 15-20 seconds per job, including operator setup time and occasionally emergencies such as bad tapes. The operation was so highly automatic that an operator was needed only to start the batch and to take it off the computer. This level of automatic operation is not usually attained in the case of the standard batch-processed monitor; it is likely that if a group of instructors had not made special efforts over the last few years to "tune" and streamline the system operation for the class that the time required for the batch-processed group would have doubled.

On the other hand, there are clear avenues of improvement in the time-sharing techniques used. The figure for "time spent in test program" shows that close to three minutes per student were spent loading and testing his program. Loading has been measured at about 15 seconds, including library search, thus on the basis of five loads (the number of assemblies) per student, 75 seconds were spent in loading. Later timing experiments have shown that if a complete library is provided, negating any need for a search of the system library, seven seconds can be saved on each loading, reducing the total time per student by about a half a minute (again assuming five loadings). The remaining minute and a half of testing time represents an average of about 45 FARDEC requests per student. By judicious reprogramming of the testing program to chain from one test case to the next on successful results, this number of requests can probably be lowered to an average of 20, saving almost a minute. (If interaction were foregone, and post-mortems given instead, the testing figure could probably be dropped to well under half a minute.) Thus slight technical

improvements on these two points would bring the average time per student from five and one half minutes down to about four minutes.

Given these figures of about four minutes for time-sharing and two minutes for batch-processing for complete solution of a typical student problem one can make a more reasoned comparison in the light of the above comments on "equivalent jobs". It should be clear, by the way, that a more sophisticated class problem, which requires several seconds of computer time for execution will add little to these figures, since execution time of the student's program is almost a negligible part of the total computer time used. Put another way, the difference between the four minutes of time-shared time and two minutes of batch-processed time represents the maximum extra overhead involved in processing student programs which do negligible computation. If the student programs required, say, minutes of execution time extra overhead as a fraction of total time used would be considerably smaller, since the execution time would add approximately the same amount to each figure.

2. Console Time used. Several interesting observations can be drawn from the statistics of console time used. First of all, although students were asked to sign up for half-hour periods at the console, the average time used in a session was in fact 35 minutes. A number of students were quite conscientious about logging out at the end of thirty minutes, but several took advantage of free consoles and continued to work for a little while longer; if the thirty-minute restriction had not been arbitrarily imposed, probably average times would be a little greater. It was noted by several students that the first period at the console particularly could profitably be extended to permit slow typists to at least get to the point of testing their program once. A more reasonable sign-up procedure might be to break each hour into three 20-minute segments, and have students sign up for one or two segments depending on the amount of input they have to do. In most cases, they would sign up for two segments for their first session, and only one afterwards.

There is no doubt, of course, that the average console time used, 102 minutes, was affected by the 2-hour time limit. However, the interesting figure is the number of students who were able to get their programs working within the time limit, 24 out of 30. From a pedagogical point of view, and considering the point of the time-sharing system, probably a time limit on console time is undesirable. An alternate approach might be to give
unlimited console time and establish a grade on the amount of console or computer time used, thereby minimizing the penalty on careful, cautious thinking. During most of this experiment, however, console time was in fact a precious commodity; this situation is not likely to change in the near future. As a practical matter, the two-hour limit probably only encouraged thrifty use of console time, and appeared to have only a minor effect on whether or not the student finished the problem.

One other problem should be mentioned with respect to amount of console time required. This problem is "computer-down" time. On the first day of the experiment, for example, the time-sharing system did not come into operation until 5 p.m. in the afternoon (instead of 9 a.m. as scheduled). This problem was anticipated, however, and students were scheduled to work only 10 out of the 14 time-sharing hours of each day. Thus on the two weekends 40 out of a possible 56 hours were scheduled leaving 16 hours for emergencies. In fact, the system was not in operation for 12 hours during this period, but the only effect this had on the experiment was to require rejuggling of signup times. (Also, the students had standby privileges on other teletypes; when other users were not active students were permitted to use these on a first-come, first-served basis. At times, then, there were three or even four students working simultaneously rather than the scheduled two.) It is likely that if tight scheduling had been attempted the experiment would have ended in failure.

With these considerations in mind, some general scaling figures may be constructed. The general rule observed is that to allow for emergencies only about 2/3 to 3/4 of the available time should be scheduled, and a student will require around 2 hours to solve a simple problem. (A later problem could be more difficult and yet require no more time, since the student is now familiar with the facilities.) Thus if one console is available for 78 hours per week, only about 54 hours should be scheduled; about 27 students could complete a problem in the course of a week with this facility. Any more specific statements as to the number of consoles required for a given class would of course require knowledge of the number of problems to be solved, and the desired rate of solution; for example, one problem could be stretched out over three weeks, allowing 81 students to participate with the one console described above.
Note also that a console is required for the instructor during the time that the student is having his program graded. The system was designed for a situation in which say, 25 students are working simultaneously, and the instructor's program has something to do a substantial portion of the time. The system could easily be modified to collect student solutions and have the instructor log in to grade them say once a day, rather than immediately.

One final result should be noted, that is student reaction to the experiment. Among the statistics collected were answers to two questions about student’s impressions. The results of both questions were practically unanimous. The students did not feel that learning the time-sharing console language interfered with their debugging, and they much preferred the time-sharing method to the batch processing.

These reactions were, of course, affected by the novelty of using a time-sharing system, but the pedagogical effect of student enthusiasm for the facilities should not be overlooked.

VII. Conclusions: In brief, then, these are the results of the experiment:

1. Students using the time-sharing system reached a similar level of success in getting their programs debugged as did those using batch-processing.

2. The time-shared group used an average of 5 1/2 min. of 7094 time apiece; suggestions have been offered to reduce this time to about 4 minutes. The comparison group used about 1 minute of computer time on a highly tuned monitor system, comparable with about 2 minutes on a more typical batch-processing system.

3. Two hours of console time per student was sufficient for most of them to complete their work. In scheduling a console, however, only about 2/3 to 3/4 of the available time should be scheduled, to allow for computer down time and similar emergencies.

In addition, the general response of the students to the experiment and to the time-sharing system was enthusiastic; this factor should not be overlooked.
VIII. Acknowledgements: Most of the programming required to carry out the experiment, including minor modifications to the FARDEG command, was done by R. H. Campbell. In addition, M. G. Smith, R. S. Fabry, and J. S. Zucker contributed time on weekends to act as instructors for the class. The Computation Center Programming Staff answered innumerable questions about their time-sharing system. In general, the 6.41 students were patient, understanding (indeed, enthusiastic) subjects of the experiment.

Submitted June 15, 1964

J. H. Saltzer

Instructor, Department of Electrical Engineering
APPENDIX

Special notes written for 6.41 students for the time-sharing experiment.

1. Preliminary announcement
2. Description of the time-sharing system.

(The pages are numbered as they appeared in the 6.41 notes.)
SPECIAL NOTES ON MACHINE PROBLEM #5 (MAX)

As an experiment, roughly half of the 6.41 class will do machine problem 5 in the usual manner, turning in decks in 26-167; the other half of the class will check out their solutions at consoles of the time-sharing system. Names of those who will do the problem with time-sharing are posted in 26-167; the rest of this set of notes is directly only at that group of students.

The first major difference in solving a problem with the aid of the time-sharing system is that you do not have to keypunch a deck of cards. Instead, you type your program directly in at the time-sharing console. You should therefore, work out your solution to MAX in the usual fashion, and plan to type it in to the computer during your first session at a console.

Time-sharing consoles will be available in Room 26-203 from 8 a.m. to 11 p.m. (except for an hour from 4-5 p.m.) on the Saturdays and Sundays, May 9, 10 and 16, 17. You will be able to sign up for up to four non-consecutive half-hour sessions during these two week-ends. It is expected that two consoles will be available at all times, and sign-up sheets will be posted for them in 26-167. If other consoles are free, you may be able to use them, but you are limited in any case to no more than 2 hours total console time.

The time-sharing system write-up provides details on which buttons to push at what time, but you will probably have to experiment a little at first to become familiar with the console and the system programs which help make it useful. Keep in mind that you are participating in an experiment (which may fail). If you are puzzled by any aspect of the time-sharing system an instructor will be available to answer questions and explain mysterious comments made by the system.

The testing program for MAX which will be available at the time-sharing consoles has 10 test cases built into it numbered 1 through 10. These numbers are used with the "g" request described in the time-sharing system writeup.

Please save all your typewriter output, as it will be needed in the collection of statistics on the experiment.
The compatible Time Sharing System (CTSS) allows the user to interact quickly with the computer. This interaction should prove very valuable if you as you work on your next machine problem.

Debugging and testing a complex problem with the time-sharing system involves the following four steps:

1. Assuming you have written a solution to the problem in the form of a FAP language program, you must type this program in an electric typewriter or electric line printer. (The operation of these machines is described in pages 101-120.) You will have to follow the instructions of your FAP program, which will be embedded and placed in a file.

2. After typing your program, you will translate it. For this purpose, the FEMTRAN Assembly Program (FAP) is available. This assembler will read your symbolic program and produce a machine file containing the machine language version of your program. If this step fails because of syntax errors in your program, you can edit necessary corrections into your symbolic program file.

3. When you have successfully assembled your program, you will proceed to test it. Here you will use a powerful debugging program named "FAPDEBUG" which permits you to "inverse" your program. You will see any bugs in your program, correct your symbolic program, and return to step 2 to try to translate the new version.

4. After you are convinced that your program works, you will submit it to your instructor for grading.
General Information:

The steps outlined above are completely under your control when you use the time-sharing system. You determine which of the steps should next by typing one of several commands to the time-sharing system. For example, you can ask that a symbolic program in a file named S001 be translated by TAP by typing the command:

TAP SMITH

The commands described below are extremely useful, and you should be familiar with them before beginning your first session.

LOGIN

The LOGIN command is given immediately after dialing the computer (see page 119 for the two dialing procedures). This command enables the supervisor program to identify you and to keep records.

You type:

LOGIN E3679 SMITH

where E3679 is the problem number assigned to 6.41. Remember to substitute your own last name for SMITH.

The computer will then type out

PASSWORD

and you should respond by typing the number which will be given to you by your instructor. The passwords are kept secret to prevent unauthorized persons from using CES. (The password which you will be given is only good during the scheduled hours.) If you follow the above directions, acknowledgments will be given of your LOGIN, and the letter "M", followed by the numbers will be typed to indicate that you may proceed.

Before you begin using the time-sharing system you must have written a TAP program to solve the assigned machine problem. With the time-sharing system you will debug your program and have it graded.
The first step is, of course, to get the program into the computer. The concept of a file should be expanded slightly here. When you allocated a certain amount of space in the disk storage, you created a disk file cabinet. You may create files by typing characters into it and place them on the "file drawer." When creating a file you must assign a name to the file so you can identify it later. By convention you file name has a two-word name. The second word of the name is usually the language the program is written in. In the example, the first two-word file name would be SMITH FAP

Typing Errors

It is considered normal for a typist to make occasional mistakes. There are simple conventions apply to all errors. Errors. Error return is the signal that you are satisfied with a line. Unless using the carriage return, however, you may make corrections in the file if you have entered a wrong character and notice it in time. If you make a mistake, press the delete character signal. This will cause the characters to the right of your error to be deleted. You may then correct your error by typing the correct character and pressing the carriage return signal. The whole line is immediately a new line on the computer's card.

INPUT

The INPUT command is used for inputting new program data into the computer. Type the command:

```
INPUT
```

The computer will then begin generating line numbers and will be synchronized with the lines you type. The first line will be numbered 0. The second line will be numbered 0 + 1. You type your program, one instruction per line.

A computer is needed to support this section of "A simple..." If you read our example carefully, you will find that the example is typed on cards with the location symbol "and a column 1-9 operation code in each column 9-10. This explains it.
spaces which are hard to count) you may see the number 1234, marked "1234"
FEED" on most teletypes) to move from one "field" to the next. In the
example here is represented by the character 1234 in line 1. A
similar FAP instruction might appear as follows:

0070 LIST -> 0100 -> 0130 -> SHORT STARTING LINES

In this line the number 0070 was typed by the FAP command everywhere.
else by the typist.

The input command has two modes automatic and manual. In automatic
mode line numbers are generated in sequence by the computer after each
carriage return you type. If you have typed a line incorrectly and have
given the carriage return, you may make a correction by changing the manual
mode

To enter manual mode from automatic you type a carriage return at the
first character of a line. Note that a space character typed as a
first carriage return will not change the mode, but will present a blank line
(which may be dismissed) in your program.

Note the initial carriage return is given, the computer will continue
typing the word

1234,

after which you must type a line number, followed by a blank and another
line number that has already been used. The new line will replace the old
line with the same number. You can insert lines between already used lines
by specifying an appropriate intermediate line number.

To return to automatic mode, merely type a carriage return after the
computer types 1234.

Thus, if you are at line 100 and you wish to insert two lines between
lines 70 and 100, you might type the following.
EXPLANATION. Being initially in automatic mode, the computer typed '00120' expecting a line. An immediate carriage return switched to manual mode, and the computer typed 'MAN'. The typist then typed a line number himself '72', followed by the first line he wanted to insert. The carriage return at the end of the line caused the computer to type the second 'MAN'. After typing the second line, the typist gave a carriage return. Waiting for the 'MAN', then typed another return, putting him back into automatic mode at line 120.

When you have finished typing your program, you should type the first give an extra carriage return to enter manual mode, then type the line

MAN FILE SMITH FAP

substituting your own last name for SMITH. The letters MAN were typed by the INPOL command program.

PRINT

To make certain that you have typed everything correctly, you may wish to print out your file. To do this, type the command

PRINT SMITH FAP

EDIT

If you wish to change your program now (or at any later time) you must make use of the EDIT command.

By typing the command

EDIT SMITH FAP

you can return to the INPOL command with file SMITH FAP. The computer will assume you wish to add lines to SMITH FAP, and type out a line number. You will probably want to make changes to your program by going into manual mode.
When you have made the required corrections, file your program as before by saving

FILE SMITH FAP

while in manual mode. The corrected program will restore the old version.

FAP

Having typed in your program, and possibly made corrections, you are now ready to begin checking it out. The first step is to translate it into machine language. For a FAP language symbolic program, a type

FAP SMITH

will success in its attempt to translate your program. It will create a new file, SMITH DBS. If unsuccessful, FAP will type the erroneous instructions with a single letter at the left indicating the nature of the error. These letters and the errors they indicate are described in the FAP manual (page 2). If you have made a mistake, figure out how to correct it and do so with the EDIT command as described before.

RESUME EDIT

Testing your program:

A special testing program has been prepared to help check out your subprogram. You can obtain this testing program by typing the command

RESUME "TEST DB 300"

The testing program will load your program, prepare for testing. Give a command then wait for further instructions from you. When it is prepared, you can give it two or several "requests." These requests are usually given by typing a single letter, perhaps followed by a parameter. The first request is:

1) Go to test. (The first request you will make is "1" or "2")

2) Quit testing. I want to make a correction.
If you have requested that testing proceed, the testing program will return full information about the test case it is supplying, then transfer to your program. One of several things may happen now: With a little luck, your program will return to the testing program with some answer. The answer will be printed and the testing program will wait for another request. If this didn't happen, type the command PARMED to get back to the testing program.

If your answer is correct, you will probably want to make another request to go on to another test case. If the answer is wrong or you have to type PARMED, you will want to look at various locations in your program to see what was placed there by your program. This is the point where the time-sharing system differs most from the batch-processing method. You can obtain information about your program, and immediately proceed with testing. Usually the greatest problem is figuring out what questions to ask. The following requests are among those you may use (SM is any symbol in your program):

3 SM The contents of location SM in your program will be typed out in detail. If you then type a carriage return, the contents of location SM+1 will be typed out.

3 SM The contents of location SM in your program will be typed out in the format of a COBOL instruction with symbolic addresses. No sense carriage return convention applies.

1 SM The contents of location SM in your program will be typed out in the form of a decimal integer. The sense carriage return convention applies.

In place of SM in the above requests, you may insert an expression such as "$164" or an absolute octal location such as "1078". The following special symbols may be used in place of "SM":

$HI, $MN, $S, $X1, $X2, ..., $X7.
When you have thoroughly checked out your program and are convinced that
it is working properly, you may submit it to your instructor for assignment
of a grade. You are allowed to do this only once, so make certain that the
program you submit is your best effort, and is working correctly. You
submit your program by the following command:

RESUME GRADE SMITH USS

This command will take from three to five minutes, at which time a result of
the grading run will be typed out on your console and you will be logged out

General Tactics:

Since you have a limited amount of console time, which you may apply to
this problem you should use it wisely. If you run into trouble and cannot
figure out how to proceed, you should log out and obtain help from your
instructor. You may log out at any time by typing the command LOGOUT.
When you log in again you will find all your files intact and you can
continue from where you left off, if you desire. It is probably best to
spend no more than 30 minutes at a console at a time. You will find
yourself wasting time if you say longer "trying things" to get out of some
predicament.

Dial Up Procedure:

The computer interconnects with typewriter consoles via a telephone system.
The first step when using the time-sharing system then, is to dial the computer's
telephone number. The exact dial up procedure depends on the type of console
you are using.

TELETYPewriter

Below the dial is a row of buttons. Depress the one labeled
"ORIG," and turn up the speaker volume. When you hear a dial
tone, dial 261 or 251. After the teletype stops making
strange noises it is ready to use.
105d samples These are the older ZIP type samples. First, place in the upper
storage. Ask someone to show you where the start button is. Pick up the telephone group. Lay it on the table so that the
bell is a flat pane and flat. Now, while the phone is on
hand, then depress the 'data' button for the first time.
Zizing up. When the prompts light, press the next menu
to type.
STATISTICS COLLECTED FOR 641 TIME-SHARING USERS

Attached to this sheet are four statistics collection worksheets and a statistics summary sheet. Write out these sheets by referring to the typewriter output produced while you used the time-sharing system. Your grade for machine problem 111 will be recorded until you have returned the statistics sheets.

Four different kinds of statistics are being collected:
1. Number of minutes you spent at a console
2. Total number of minutes of computer time you used
3. Number of times you used each command.
4. Total amount of computer time spent doing each command.

Starting with the four worksheets, allot one worksheet to each login/logout period. (If you need more than four, the secretary has extras, or provide a reasonable facsimile of your own design.) Then, go through your typewriter output from each period, filling in an entry for each command you used. The times associated with each command are the numbers typed after the letter "t" when the command is finished. For example, when you used FAP, your listing might look like this:

FAP SMITH
W 1011.4
42 LE 501 FIRST LOCATION NOT USED BY THIS PROGRAM.
R 8.400 7.150

In this case, the command was typed in at time 1011.4 (which is irrelevant as far as our statistics are concerned) and the time spent doing the command was 8.4 - 7.1 = 1.3 seconds. Therefore you would enter in the FAP column "1.3". The time spent in an INPUT or EDIT command is included in the time typed following completion of a "FILE" command. The time spent in
RESUME TEST is typed out when you type "q" or when your program goes wild, causing you to have to type "FAPDBG". The time spent in RESUME GRADE cannot reasonably be measured, so don't attempt to put down a figure for that command. Note that although you spent minutes (or hours) at the console, you never used the computer foremore than a few seconds at a time, and probably used only a total of 3 or 4 minutes of computer time.

After filling in the separate entries on the worksheet, add up the times in each column, enter the total, then count the number of times you used that command and place the count at the bottom of the column. In the case of INPUT-EDIT-FILE, the sequence INPUT-FILE is one entry, as is the sequence EDIT-FILE. The next to the last blank on the worksheet, "Computer Time Used", should be the number typed by the "LOGOUT" command.

When all the worksheets are filled in, calculate the necessary totals and fill out the summary sheet. Calculate the "Total Computer Time Used" by adding up the times typed following "LOGOUT" commands, rather than totalling separate command times. (The answer will probably be different.)

Return the filled-in statistics sheets to the secretary, Miss Bearlstein, in Room 26-260A.
6.41 Time-Sharing Statistics Collection Worksheet.

**Time of LOGIN:** ___________ **Date of LOGIN:** ___________

Times used, in seconds. ___________ (sum of figures typed following "R")

<table>
<thead>
<tr>
<th>INPUT: EDIT, FAP</th>
<th>RESUME TEST or FAPDG</th>
<th>PRINTF</th>
<th>ALL OTHER COMMANDS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**LOGOUT at time:** ___________ **Date:** ___________

Computer time used: ___________ minutes.

Console time used: ___________ minutes.
6.4 Time-Sharing Statistics Summary:

1. Total console time used: __________ (minutes).

2. Number of distinct console sessions: __________

3. Total computer time used (sum of times typed by "LOGOUT" commands): __________ minutes.

4. Time spent in INPUT, EDIT, and FILE commands: ____________

5. Number of INPUT--EDIT--FILE entries: ____________

6. Time spent in FAP commands: ____________ seconds

7. Number of uses of FAP command: ____________

8. Time spent in RESUME TEST and FAPDBG: ____________ seconds

9. Number of uses of RESUME TEST and FAPDBG: ____________

10. Time spent in PRINTS: ____________ seconds.

11. Number of uses of PRINTS: ____________

12. Time spent in other commands: ____________ seconds.

General Questions:

1. Did you find that learning the time-sharing console language interfered excessively with your real job, learning about FAP?

2. Given a choice of doing a future problem with time-sharing or with the standard method, which would you prefer? Why?

3. Comments and Suggestions: