8TRAN language and the PDP-8 facility at the L. L. Thurstone Psychometric Laboratory*

THOMAS S. WALLSTEN†

University of North Carolina, Chapel Hill, North Carolina 27514

Research in the L. L. Thurstone Psychometric Laboratory includes the study of inductive problem solving (e.g., Johnson, 1971), multidimensional scaling (e.g., Young & Cliff, 1972), individual, multistage decision making (e.g., Rapoport, Jones, & Kahan, 1970), and revision of opinion (e.g., Wallsten, 1971). Virtually all of the experiments related to this and other work are administered on-line with our PDP-8 computer facility (see Jones, Johnson, & Young, 1969). As a result, the demand for computer time is great, and it is necessary to run experiments and construct programs as efficiently as possible. To this end, the laboratory has dedicated substantial effort to the development of appropriate software systems which are sufficiently general to have a wide range of application. Two of these will be reported here.

One is the Remote Access Time-Sharing System (RATSS) developed by Mr. Marvin Conrad (see Jones, Johnson, & Young, 1969). In most of the experiments conducted in the laboratory, a considerable amount of time elapses between presentation of the stimulus and initiation of the response (i.e., at least 1 or 2 and sometimes more than 30 sec). The computer operates in terms of microseconds and normally is idle during most of this response period. RATSS, however, allows this idle time to be utilized so that up to eight independent experiments may be run simultaneously.

RATSS requires certain hardware besides the basic PDP-8. These are an additional 4,096 words of core memory, 4K of disk storage for each user, an extended arithmetic element (for hardware multiply and divide), and a sufficient supply of Teletypes.

*The developments reported in this paper are the products of many people. Chief among them are Mr. Richard Helwig, who has provided close support and critical advice on all the projects, Mr. Arthur Coston, and Mr. Len Stroble. The work has been supported by Grant GU-2059 from the National Science Foundation to the University of North Carolina and by Grant MH-10006 from the National Institute of Mental Health. The author would like to thank Mr. A. Coston, Mr. R. Helwig, and Dr. E. S. Johnson for critically reading an earlier draft of this paper.

†Requests for reprints should be sent to Thomas S. Wallsten, L. L. Thurstone Psychometric Laboratory, University of North Carolina, Chapel Hill, N.C. 27514

Additional memory is convenient, either in the form of disk or magnetic tape.

Earlier in this conference, Uttal (1972) argued against the use of time-sharing systems. One of the reasons was that a second (and even a third) computer could often be purchased for the cost of the necessary extra hardware. The point is undeniable but fails to meet the fact that many experiments and much software development would be impossible without the additional devices. Thus, when necessary, we have a single rather powerful system (equipped with other hardware, e.g., scopes, besides that described above), and, other times, we have what is effectively up to eight independent computers.

When RATSS is used, each S sits in a private cubicle at a Teletype which is employed both to present stimuli and as a response device. RATSS operates in such a way that the computer acts on the S's response with time delays in multiples of .6 sec. Thus, the S normally has to wait 0 or .6 sec for a new stimulus to follow a response. Rarely does he have to wait 1.8 sec or more. These lags have never presented serious difficulty in running experiments.

Programming for experiments to be administered under RATSS is no different from programming for a single user system, except that special input-output (I/O) statements are needed. Assembly language is used, and when the program is completed, debugged, and compiled, the object code is stored, in our case, on magnetic tape.

Then to actually run an experiment, the program is called separately for each S at his Teletype. RATSS maintains appropriate I/O buffers for each S and coordinates communication between him and his program. Thus, I/O may go on simultaneously for all Ss, although a single user program resides in core at any instant.

Briefly, RATSS operates as follows: it resides in the first 4K of core, handles all I/O, and schedules the flow of users' programs into and out of the second 4K. Assume that an S's program is in core and has control of the computer. It receives the S's latest response from RATSS, computes the new stimulus, and issues a Request I/O

command. Typically, this tells RATSS to print the new stimulus on the Teletype and to receive the new response, and this shifts control back to RATSS. RATSS, in turn, displays the stimulus, obtains information from the program needed to accept the response, and turns control back to the program for further computation, if necessary.

Usually there is very little to be done, since input from the S is needed for the next step. The program thus enters a WAIT state, and control shifts back to RATSS, which checks its READY STACK to determine whether I/O from any user is completed. If so, that user's program is ready to resume computation. Therefore, the previous program is copied onto disk, and the new one is read into core and given control.

The time-sharing system has greatly facilitated the actual running of experiments, but it has not ameliorated other limitations associated with the on-line use of the computer. For example, programming a moderately involved experiment or series of experiments in Assembly language is not easy, even for experienced programmers. Often several months are required for the task.

Also, it is not easy to handle data records, i.e., to set up and manipulate data files (although RATSS has eliminated the burden of actually performing the I/O). Another limitation is that generally I/O must be restricted to specific formatted forms. Finally, the range of possible experiments is somewhat limited. In particular, alphabetic character manipulation and evaluation is very difficult. This facility would be desirable, for example, in work on problem solving, decision making, memory, and psycholinguistics.

These and other considerations have led to the development of a high-level compiler language for the PDP-8 called 8TRAN. The language has been developed mainly by Mr. Arthur Coston. 8TRAN can run on any PDP-8 which has an extended arithmetic element and can support the PS/8 monitor system. However, it is monitor-system-independent at run time. That is, after a program has been compiled and the object code made available, it can be run under any monitor.

The structure of 8TRAN is very much like that of FORTRAN; with minor changes, most FORTRAN programs will compile and run in 8TRAN. However, the present language is considerably more powerful than FORTRAN and includes many features found in PL/1, as well as some not found in most

languages. A few of these will be described.

In addition to the usual data forms (including single- or double-precision integers and floating point), 8TRAN provides for bit strings of fixed length, character strings of variable or fixed length, and pointers. Pointer operations often permit programs to be considerably smaller than they would be otherwise and allow the possibility of list processing.

Input and output may be formatted but need not be. Default formats exist for output, and free form may be used for input.

Character manipulation and evaluation, including string extraction and comparison, are done as easily as are arithmetic operations. Comparison, concatenation, and Boolean operators are available, as are the other operators normally found in FORTRAN. Mixed-mode expressions may be used freely.

These features alone provide great flexibility in programming and therefore in experimental design. Others have a similar effect in terms of data storage. 8TRAN permits the user to define INPUT, OUTPUT, or UPDATE files, the latter being used for both input and output. The files may be addressed either in stream (ASCII characters) or in record (internal format) form. Thus, it is as easy to write, say, from core to disk as it is to receive input from a Teletype.

8TRAN makes very efficient use of core storage, something especially desirable with a small computer. Thus, the linking loader puts together only those subroutines that are required by the program. Temporary buffer storage is allocated dynamically during the execution of a program.

Although this description of 8TRAN is far from complete, it should provide some indication of its power and potential usefulness. Experience in our laboratory has already shown that when 8TRAN is employed, program-development time is a fraction of what it would be otherwise. Also, 8TRAN is considerably easier to learn than is Assembly language, especially if one has any familiarity with FORTRAN or PL/1. RATSS is currently being revised and generalized so that it may be one of the several monitors that can support the data-management concepts of 8TRAN.¹ Thus, we feel that the range of empirical applications of the PDP-8 has been greatly enhanced.

REFERENCES

- JOHNSON, E. S. Objective identification of strategy on a selection concept learning task. *Journal of Experimental Psychology*, 1971, 90, 167-196.
JONES, L. V., JOHNSON, E. S., &

YOUNG, F. W. Computer control of psychological experiments. Chapel Hill, N.C.: Psychometric Laboratory Report No. 80, November 1969.

RAPOPORT, A., JONES, L. V., & KAHAN, J. P. Gambling behavior in multiple-choice multistage betting games. *Journal of Mathematical Psychology*, 1970, 7, 12-36.

UTTAL, W. R. Misuse, abuse, overuse, and unuse of on-line computer facilities by psychologists. *Behavior Research Methods & Instrumentation*, 1972, 4, 55-60.

WALLSTEN, T. S. Probabilistic information processing, Bayes' rule, and conjoint-measurement. Chapel Hill, N.C.: Psychometric Laboratory Report No. 98, August 1971.

YOUNG, F. W., & CLIFF, N. Interactive scaling with individual subjects. *Psychometrika*, 1972, in press.

NOTE

1. It is anticipated that the revision will be completed by the time the conference proceedings are published.

GASPS: A General ASynchronous Processing System for the PDP-8*

THOMAS D. WICKENS

University of California, Los Angeles, California 90024

and

JAMES HOWARD, GLENN RICE, and RICHARD B. MILLWARD

Brown University, Providence, Rhode Island 02912

Programs for the on-line control of human verbal-learning experiments are characterized by an abundance of short program segments interspersed with variable delays. Most operations (presentation of a stimulus, receipt of a response, even generation of feedback) require no more than a hundred machine instructions, but the sequence in which these basic sections are executed is usually more complicated and often cannot be determined in advance. This uncertain ordering of events is even more evident when several Ss are run simultaneously, but asynchronously, at their own pace. It is impossible to write such programs in linear languages such as FORTRAN, and the contingencies are usually sufficiently complex as to make the state-transition languages for operant experiments unworkable. What is needed is a way to control a sequence of contingent events, i.e., a program that allows relatively sophisticated scheduling of small program segments. Our attempt to develop such a system for the PDP-8 computer has resulted in the program reported here, GASPS, a General ASynchronous Processing System.

The original GASPS program was

written to control four subject stations (keyboard, random-access slide projector, and indicator lights) from a PDP-8/S computer with a 4K memory and an ASR-33 Teletype at the Human Learning Laboratory at Brown University. The only hardware addition to the computer other than the subject stations was a 12-bit clock that could be set by the program and would interrupt on overflow. Subsequently, the system has been extended to include a video display for each S, a high-speed paper-tape reader and punch, a 64K random-access disk, and a magnetic-tape drive. Corresponding modifications have been made to GASPS, and the entire program now requires about 1,800 locations, leaving the remainder for the user's program. As noted below, the disk can be used to store program segments.

In a GASPS program, the short sections of code mentioned above are written as autonomous units, usually performing single functions. They will be referred to as tasks. Examples of tasks involved in simple learning experiments are selection of a stimulus, presentation of the stimulus, enabling the system to receive valid responses (after the stimulus has appeared), recording the S's response, presentation of feedback, generation of a record of the trial, and so forth. Control is passed to the tasks by the GASPS monitor and is returned to GASPS when the task is complete. When no task is pending execution, GASPS retains control.

Calls can also be made from a task to a number of subroutine entry points within GASPS. Three of these

*Please send requests for reprints or information to Professor Richard B. Millward, Department of Psychology, Brown University, Providence, Rhode Island 02912. A manual describing the use of GASPS in detail, with an example and a PAL listing of the program, is available from Professor Millward. Please send \$2.50 to cover duplication and mailing costs. A paper tape of the GASPS program (in PAL form) can be obtained for the same price.