COMPUTER SYSTEM SIMULATION: A DESIGN EVALUATION TOOL

Dr. Robert S. Feingold
U.S. Air Force Data Automation Agency

INTRODUCTION

The increasing complexity of weapon systems is an ever present fact of life in the military system procurement business. Complexity impacts development cost, development schedules, and operational support once the system is deployed for use. Complexity also has a significant impact on how well user stated system performance goals are met. Generally, as design complexity increases, so does the level of effort required to meet user performance needs - on time and within budget.

Weapon systems which contain embedded computers are generally among the most complex. An embedded computer system is a collection of hardware devices and software which serve to support an overall weapon system mission. Embedded computers are used to process radar track inputs, drive display devices, control propulsion systems, and supervise the launching of various categories of ordnance. On a larger scale, computers are employed by combat elements directly to process intelligence data, route command message traffic, solve complex trajectory problems and assist in controlling the movement of supplies and replacement parts to front line units.

Like hardware, computer software goes through an acquisition process consisting of concept, validation, full scale development, and production deployment phases. The major concern in the concept phase is the derivation of the computer system requirement in consonance with the overall requirements for the weapon system. The validation phase emphasizes the design effort where proposals for hardware and software configurations are brought forward and evaluated against user requirements. In the validation phase, design decisions are made which ultimately effect how well the computer system will meet its performance goals. Full scale development concentrates on the acquisition of hardware and the programming of software according to the design selected in the validation phase. Full scale development is lengthly and concentrates on testing of hardware and software in a controlled environment. The production-deployment phase starts after the production baseline is established. Software is easily reproduced for deployment while hardware must proceed through the usual manufacturing and qualifying steps.

Although the process outlined above, for the most part, follows the traditional acquisition approach, the potential for encountering major development problems still remains high. Problems usually result from decisions made quite early in the development process. Quite often the amount of effort devoted to hardware-software design trade-off studies in the validation phase is not sufficient to adequately identify potential problems. A contributing reason for not devoting resources to computer system hardware and software trade-off studies has been the lack of adequate tools to assist in this effort. Software design trade-offs are extremely difficult to study before the software is written or before the hardware becomes available. As a result, the computer system designer finds himself making hardware configuration and software implementation decisions based primarily on his past experience. The consequence may often be an imbalance in total system performance due primarily to the failure of the embedded computer system to adequately support the weapon at the desired level of effectiveness.

The primary goal of this paper is to present a computer system design evaluation methodology appropriate for the validation phase

that involves the user, the designer, and the developer in an iterative and interactive design evaluation activity. This methodology, which relies on the operations research technique of system simulation, is asserted to be well suited for use in evaluating complex embedded computer systems. In this environment, the simulation programs represent relevant system and subsystem relationships in model form and can be used to evaluate the cost, schedule, and performance impact of various proposed designs in response to user stated requirements.

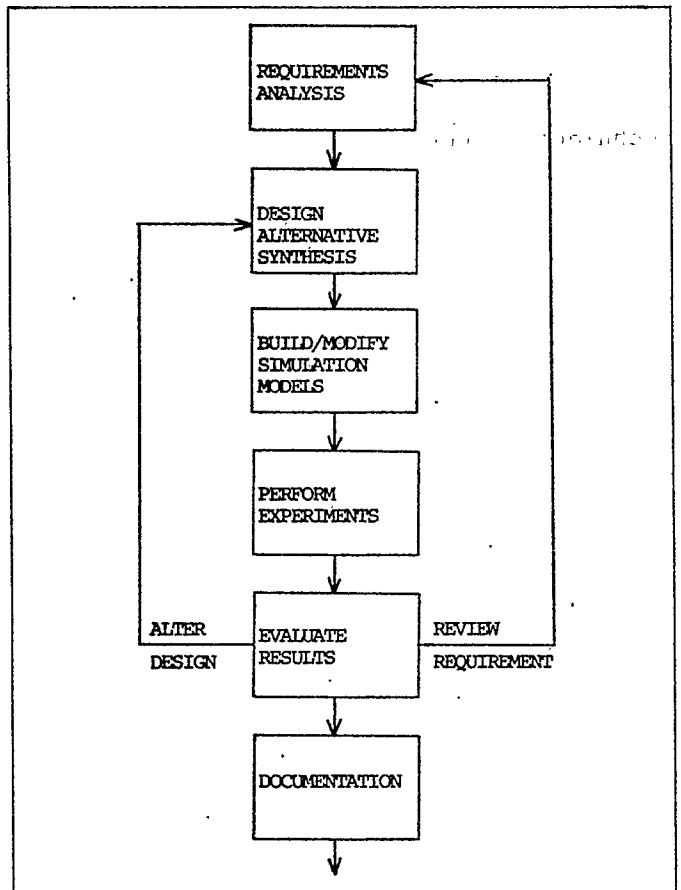
A "hypothetical situation" is used to illustrate how computer system simulation can be employed in the acquisition process. This approach is intended to dramatize the process by permitting an indepth look at the computer system simulation technique and how it can be used by a program office responsible for an embedded computer system development.

COMPUTER SYSTEM DESIGN EVALUATION METHODOLOGY

For our purposes we will view the design evaluation process as consisting of six steps related to one another as shown in Figure 1. The process is iterative in that the design evaluation teams can use the results from experiments to develop additional design alternatives and suggest challanges to existing user requirements. This iterative process, however, cannot go on forever. For one thing, time is usually the major constraint. For another, and perhaps even more important, there is a limit to the sensitivity that model based evaluation techniques can show given gross statements of requirements and design alternatives. The evaluation must stop sometime and culminate in a set of design decisions which are carried forward for implementation into hardware and software.

Requirement Analysis

Basic to the success of the design process is an understanding of the user's objective. Achieving this understanding is often difficult since the path the designer follows is often cluttered with user jargon and user design biases included within the stated requirement. To overcome these and many other similar problems, user-designer-developer teams should be formed to work through the entire process. Clarification of user intent and developer capability at an early stage usually contributes to overall success by reducing the number of false starts in the beginning and by shortening the design approval process later on.



DESIGN EVALUATION PROCESS

FIGURE 1

The output from this first step is an initial understanding of the problem the user wants solved, some insights toward developing a set of potential solutions, and, most important, an integrated user-designer-developer team that can function effectively through the remaining five steps.

Design Alternative Synthesis

Synthesis of system design alternatives is a creative process which combines the designer's understanding of the requirement and his ability based on his training, knowledge of the technology, and past experience. Some alternative evaluation takes place at this step in the process. Realistically it is not desirable to eliminate design evaluation totally at this point. Although many proposals will be made, only a few will be worth further study. Judgement is essential to filter out those proposals which do not merit further considerations. In the environment being discussed here, this judgement is supplied collectively by user representatives working together with system design and development specialists.

In the past, all design evaluation was accomplished using techniques which did not always account for the inherent complexity of modern computer systems. In many cases it could not be helped since economical evaluation techniques were not available for doing indepth analysis nor were there techniques sufficiently responsive to the designer's need for quick and accurate answers to design trade-off questions. As we shall see later on, the computer system simulation approach is both available and responsive.

Model Building

The process of model building satisfies two very critical requirements. First, it permits the user-designer-developer team to articulate their combined knowledge of requirements and design alternatives into a physical entity, called a model, which is understandable and can be subjected to intensive analysis. If a model cannot be built which is easily understood and incorporates the major design features proposed in the synthesis step, then the entire process up to this point is not successful.

The second major requirement satisfied during model building is the development of a responsive tool which is useful throughout the remainder of the process. The model, if constructed properly, extends the intellectual abilities of the design evaluation team by providing them with insight into system component relationships rather than with only masses of numbers to analyze. The design team can try out logical combinations of design and requirement variables to arrive at a reasonable understanding of what the proposed system will do and how it will behave under a wide variety of circumstances.

Model building is illustrated in the next section where a model written in a language called ECSS II, the Extendable Computer System Simulator II is constructed. This language is a product of the RAND Corporation and was developed for the U.S. Government under sponsorship of the Federal Computer Performance Evaluation and Simulation Center (FEDSIM) and the General Services Administration. ECSS II is being used here because it satisfies the two previously stated modeling requirements: understandability and responsiveness. ECSS II has an English-like syntax, provisions for compact descriptions of computer elements (hardware and software), flexibility, extendability, modifiability, and the provision for economical rerun. At the same time, report generation capabilities and data collection facilities are very good.

Experimentation

Experimental runs made with a model should be designed to explore as much of the design and requirement response surface as possible. An exhaustive search is impossible. However, experiments should be performed which allow the design evaluation team to first validate the model and second to examine system performance behavior relative to changes in sensitive design variables.

Model validation is a major subject unto itself. Let it suffice for our purposes to say that a model should be validated to a point where its behavior can be explained by either a model coding error or a phenomenon which the modeler can reasonably expect from the actual system.

As a design evaluation tool, the computer system simulation model is used to obtain numerical data on the performance of a proposed system given input design variables and a simulated operational environment. A series of experiments are usually run, each with logical variations made to the input parameters. At the end of the experimental phase, the outputs are correlated with these changes in design inputs and a better understanding of system behavior is achieved.

For most computer system simulation models, there are three classes of input variables. First, there are those variables which represent hardware options such as processor speed, memory size and input-output capacity. Second, there are variables associated with software design options. Included in this class are various possibilities for implementing application software, memory management techniques, and multiprogramming control schemes. Third, there are variables related to the operational environment. For example, a message handling system operates in an environment where message arrival rates and sizes are quantified and specifications for message response times are given.

Evaluation of Results

The fifth step in the process is intended to result in one of three possible decisions. If the results of the experimental runs indicate that the proposed designs are inadequate, then the team must go back to step two and try to develop new designs. Quite often a system design which on paper appears to satisfy the requirement turns out to be a miserable failure. When a simulator conclusively shows that a problem exists, the design evaluation team has no alternative but to drop back and try to develop new designs.

The second possible decision is closely related to the first but differs enough to justify separate treatment. After several iterations, it may become clear to the design evaluation team that the state-of-the-art does not provide a feasible design for satisfying the requirement. In this case, the design evaluation team might challenge the requirement by using the model to evaluate the effect of relaxed

requirement on design choice. Full user participation in this kind of exercise is mandatory. It must be emphasized that modifications made to the requirement inputs of the model are for experimental purposes, to determine the trade-off of system cost to changes in requirements.

The third possible decision represents the end product of the process to this point: the selection of a feasible system design. Using the model, the design evaluation team can select one or a small number of designs worthy of further consideration. Given such a determination, the design evaluation team can move on to the final step.

Documentation

The final step, although it may appear anticlimactic, is really quite important. The design evaluation team must report its findings which become the basis for seeking approval to continue work. It is important that this documentation be clear on a number of key points. First, the process used by the design evaluation team to translate the requirement into model inputs must be absolutely clear. Second, the documentation must clearly and completely state all assumptions that went into the fabrication of the model for all three classes of inputs mentioned earlier. Third, the model itself must be completely explained such that the embodiment of all the previously stated assumptions are clearly identified within the model representation. Fourth, the data collected from each experimental run must be defined so that a clear interpretation of numerical values can be made. Fifth, the experimental design must be described and justified. Finally, the results from each experimental run must be described in terms that clearly support the conclusions and recommendations of the design evaluation team.

These requirements for documentation may seem excessive, but it must be remembered that computer system simulation is viewed by some as suspect because full disclosures of methodology in the past were not made and it was not clear how the recommendations produced were derived. Remember, there is no such thing as "hard simulation data" upon which high dollar decisions can be made with low risk.

ILLUSTRATION

Hypothetical Requirement

To fully illustrate the use of computer system simulation in the embedded computer system design evaluation process, a hypothetical situaion is provided consisting of a typical user requirement and one possible design proposal. Although only one design proposal is discussed, it should be clear that the methodology lends itself to handling any number of design proposals.

Suppose that a tactical field activity needs to improve its capability to quickly process information on potential tactical targets and provide responsive reports to air component commands on possible tactical air missions that should be flown against these targets. Specifically, it is desired to search lists of known targets, update these data, and display those targets that have attributes corresponding to the input search arguments. Additionally, it is desired to obtain timely printed reports which display target data in a variety of ways. That is, report formats must be flexible enough to satisfy new requirements encountered in the field.

In general terms, three capabilities are required: first, an ability to input target data into the system so that it can be retrieved at a later time; second, an ability to search and update the data base for records satisfying a number of the search input characteristics; and third, a facility to provide printed reports of previously conducted data base searches. The user also desires to record all system transactions in printed form so that hard copies of all data base changes and accesses will exist.

Based on his knowledge of the environment, the user also provides additional details of his requirement. First, he defines a system encounter to be any series of interactions between a human operator and the mechanized system. For example, a typical data input encounter consists of an input stream of characters representing target data and the output is a signal from the system that these data are safely stored in the data base. Second, the user provides information on the number of encounters that the system is required to process per hour. Third, the user provides an indication of the number of characters of input associated with each encounter category. Finally, some indications as to the volume of print characters needed to complete each encounter are provided.

Table 1 summarizes these general requirements for all encounter categories. By themselves these requirements are not sufficient to develop design alternatives let alone build a simulation model. However, for the sake of illustration, we will assume that additional information exists which helps refine the user environment to a more detailed level and allows a design (and model) to be proposed. For instance, it is safe to assume that each encounter in reality is a series of man-machine inter-

ENCOUNTER CATEGORY	ENCOUNTERS PER HOUR	INPUT CHARACTERS	OUTPUT PRINT CHARACTERS
DATA INPUT	30	80-1,000	480-6,000
DATA BASE SEARCH AND UPDATE	12	40-200	4,800-24,000
REPORT GENERATION	6	30-45	96,000-144,000

GENERAL USER REQUIREMENTS

TABLE 1

actions consisting of a human stimulus input, system processing, and final output response. Further, we may assume that the system operator interfaces with the equipment through a keyboard cathode ray tube (CRT) display device. By implication, the operator must key his inputs at a given rate and examine the outputs provided on the CRT at a certain reading speed. Also, the operator must spend some time thinking before keying in the next interaction stimulus. Finally, it is safe to assume that the data base is stored on some random access device, say a disk, which is accessed a number of times to satisfy each man-machine interaction. Table 2 provides the numerical values associated with these detailed assumptions.

Hardware

The system will consist of a central processing unit with memory, a multiplexor interface channel to the keyboard-CRT devices and the printer, and a burst mode interface channel to the disk devices which will hold the data base. Since the requirement says that the system must consist of state-of-the-art components, the capabilities of each subsystem will be restricted. So, the design proposal that we will consider is configured as shown in Figure 2. The names given to each class of components appear later in the model. Also appearing in the model are the device characteristics shown with each system component. The devices called C.PATH, P.PATH and D.PATH are names given to entities over which simulated data flows. For example, C.PATH serves the CRT to CH1 path while D.PATH serves the DISK to CH2 path. These too are used in the model.

Software

Since hardware in a computer system is of no use without software to drive it, the software too must be designed with appropriate detail. Again, some assumptions are to simplify the illustration. First, no consideration is given to operating system overhead. It is assumed that overhead is accounted for within the application software. Second, each major category of input is served by a different program which must be in memory for the interactions between man and machine to take place. Third, the system supports multiprogramming. Finally, there is a separate program that produces hard copy print from files stored on disk. Under this arrangement each program that produces print places it on disk for later printing.

More elaboration is required for each of the processing categories. First, the operator receives the material he must work with. This might be new data to input, information for searches or updates, or parameters for a required printed report. The operator logs on and waits for the system to give him access to the necessary system resources. Next, he engages in a series of machine interactions until his encounter work is complete. A request is then made to produce all printed products and the operator logs off. Since it is likely that additional encounter material is waiting, the operator is likely to become busy again.

The software process supporting the Data Input encounter category is equally straight forward. First, the logon is performed and the operator is notified to proceed. Then each time the operator provides inputs, the program places the data on disk, updates the data base index, and displays a response on the operator's CRT. When all interactions are complete, the logoff process is performed.

The Data Base Search and Update encounter category process is more complex. After performing the logon process, each operator input is scanned and used to control the search and update of the data base. Once again, when the interations are complete, a logoff is performed.

The Report Generation encounter is a simple process category. At its heart the program interprets operator inputs, retrieves data from the data base, organizes these data into the appropriate report format, and stores the report on the output spool file for later printing.

One final process, the Print Spool Program, completes the software program set. It is assumed to operate as a batch or background process and is general enough to serve a system with multiple printers. We will see that this comes in as a handy design feature later. Accordingly, the program requests a printer from the system and one by one prints the lines contained in the output spooling files.

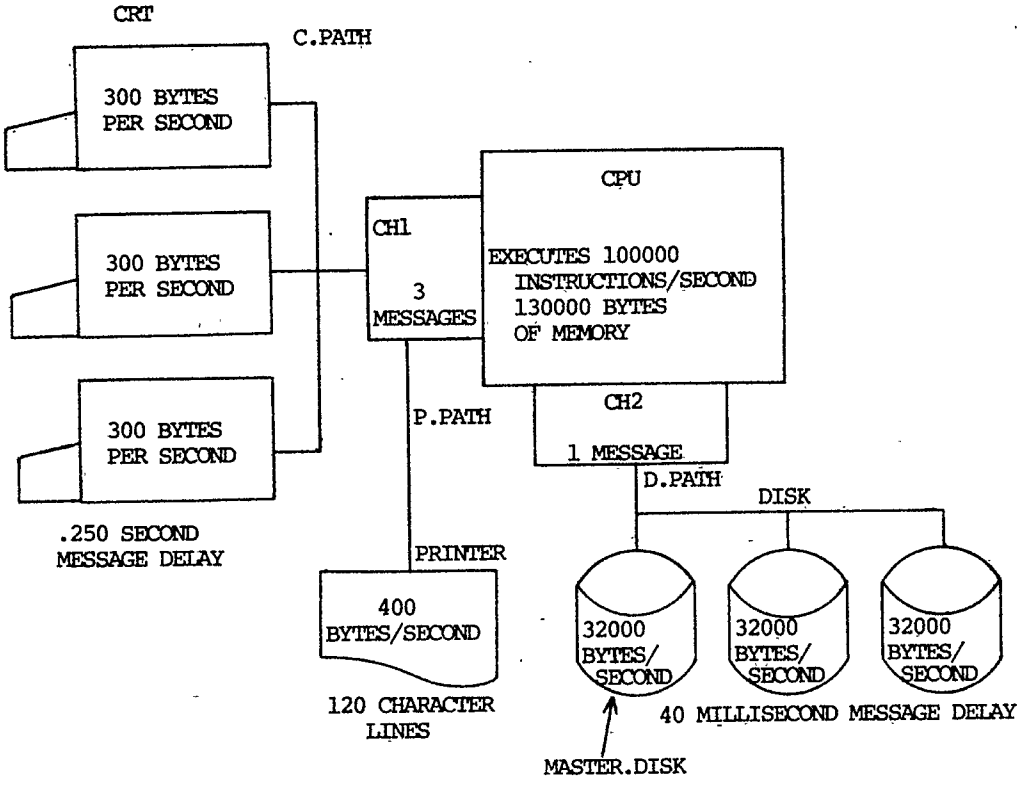
Initial Timing Study

Now that all the processes are described,

ENCOUNTER CATEGORY	REPETITIONS PER ENCOUNTER	AVERAGE KEY STROKE TIME (3 CHAR/SEC)	RESPONSE READ TIME (500 CHAR/ SEC)	THINK TIME (SEC)	DATA BASE ACCESSES
DATA INPUT	2-25	26-333	10-120	10-125	4-50
DATA BASE SEARCH AND UPDATE	2-10	13-66	5-24	15-70	100-1,000
REPORT GENERATION	2-3	10-15	4-6	10-15	40-120

QUANTITATIVE ASSUMPTIONS

TABLE 2



SYSTEM DESIGN CONFIGURATION

FIGURE 2

we can make some initial timing estimates to determine the response time characteristics of the system. To accomplish this we must make a few more assumptions about instructions executed per program as well as input-output requirements for each program. The assumed instruction counts are given in Table 2. These numbers represent our informed estimate on what is required to accomplish each process.

ENCOUNTER CATEGORY	LOW ESTIMATE	HIGH ESTIMATE
DATA INPUT	19,980	48,500
DATA BASE SEARCH AND UPDATE	1,057,200	10,156,200
REPORT GENERATION	565,200	842,800

PROCESS INSTRUCTION COUNTS
(INSTRUCTION/ENCOUNTER)

TABLE 2

Experience and knowledge of commonly used algorithms forms the primary basis for the low and high estimates. The reader should not be too concerned at this point as to where these figures originated. The model, presented in the next section, will clearly show all instruction execution steps. For simplicity, each process includes the instructions needed to complete the required printing. Table 3 shows the low and high estimates for input output times. It is assumed that within a process there is only one input-output (IO) performed at a time. Furthermore, IO is never overlapped with instruction execution. The values in Table 3 includes all IO for each process. Again, IO associated with printing is included with each of the three processes. The precise flavor of the values shown in Table 3 results from expressing IO in seconds rather than in characters per encounter.

ENCOUNTER CATEGORY	LOW TIME ESTIMATES	HIGH TIME ESTIMATES
DATA INPUT	3.194	36.685
DATA BASE SEARCH AND UPDATE	21.83	81.485
REPORT GENERATION	243.808	368.925

INPUT-OUTPUT TIMING
(SECONDS/ENCOUNTER)

TABLE 3

To complete the timing study we must account for the operator think time that is not overlapped with any other machine process. Since this can be quite complex, it will be simplified by assuming that for the three interactive processes, think time is 20 seconds, 20 seconds, and 15 seconds respectively. In summary, the average timing estimates shown in Table 4 are obtained by finding the mean from the low and high timing estimates. This is reasonable if we assume that instruction counts, data base accesses, and transaction character sizes for IO are all drawn from uniform distributions with the low and high values given in Tables 2 and 3 as parameters.

TIMING ELEMENT	DATA INPUT	DATA BASE SEARCH & UPDATE	REPORT GENERATION
EXECUTION TIME (1000000/SEC)	.342	56.067	7.040
INPUT/OUTPUT TIME	19.939	51.657	306.366
THINK TIME	270.0	120.0	25.0
TOTAL	290.281	227.724	338.406
AVERAGE RESPONSE TIME		285.470	

TIMING SUMMARY
(AVERAGE SECONDS/ENCOUNTER)

TABLE 4

Analyses, such as the ones performed here, are quite common and for simple systems are adequate as well. Still one might be suspicious because this analysis does not account for all resources needed to process an encounter. For example, nothing in the analysis above indicates that the design value of 130,000 bytes of memory is adequate for this combination of processes. Furthermore, time could be lost by each process as it waits for resources temporarily assigned to other processes. Queues can develop in a number of places. First, there are four devices on CH1 which can only process three messages concurrently. Second, there is only one CPU and a maximum of four programs can request this resource at any given time. Third, the DISK devices can only be accessed one at a time and these same four programs can each have DISK IO requests pending. Finally, we have only one printer, yet three processing programs can be generating print outputs.

MODELING AND EXPERIMENTATION

The primary purpose of this paper is to present a computer system design evaluation methodology appropriate for the validation phase of an embedded computer system

development. This stated purpose precludes a long treatise on modeling languages, modeling techniques, and simulation strategies. However, a brief look at the ECSS II model is in order at this point.

The Model

By way of a brief explanation, an ECSS II model consists of at least four sections each performing either a descriptive or operative function in the simulation. The first section, a PREAMBLE, defines global data structures, sets which are used to implement the flow of simulated work through the system, and variables that hold parametric values or accumulate performance statistics. Defintions for each entity, attribute, set, and variable are provided in Appendix A and the complete model is given in Appendix B. To carry on without plunging too deeply into detail, you might look on the Temporary Entity STAT.NOTE (See Appendix B) as representing the encounter material. The note is created during model initialization; has an arrival time; is associated with one of the three encounter categories; and is filed in the NOTE.SET when it is created.

The second section of an ECSS II model is called the DEFINITION DESCRIPTION and is used to enhance the overall readability of the model. As will become evident later, it is much easier to refer to time in SECONDS and MILLISECONDS then in arbitrarily defined "TIME.UNITS."

Any computer system simulation model must represent both hardware and software. Accordingly, the third section, called the SYSTEM DESCRIPTION, provides details of hardware characteristics in an easily readable form. In addition, each simulated data path is defined with a PATH statement.

It is interesting to note that only those characteristics needed to describe a device are stated. For example, we are not interested in IO for the CPU, so no transmission statements are provided and the CPU does not appear in any PATH statements. The CRT and PRINTER devices provide the transmission rate value and are included on the C.PATH and P.PATH, respectively, which contain the CH1 device.

The fourth section of an ECSS II model is the WORKLOAD DESCRIPTION and is used to describe the various system processes. In ECSS II there are two kinds of processes. The first or EXTERNAL PROCESS is not associated with a device capable of executing instructions and is therefore suitable for simulating human behavior scenarios or devices external to a computer like servo mechanisms or data sampling devices. The second process called a JOB is used to model computer software.

The EXTERNAL PROCESS USER first represents the human operator in our model. From the model in Appendix B it should be clear that USER first does some initial housekeeping followed by a statement which invokes a program, called PROG, which runs on CPU. USER waits for a signal from PROG and when the signal is received, USER enters a loop which simulates the stimulus, wait for response, think process described earlier. At the conclusion of the loop, USER invokes a batch JOB called LISTOFF which prints any output files generated by PROG. Later we will see that all USER processes are created at one time.

The ALLOCATE statement is used to implement a queueing mechanism. If a CRT is available, it is automatically assigned by ECSS II and processing continues. If all CRT's are busy, ECSS II puts the USER process to "sleep" by queueing the CRT allocation request. When another USER process releases a CRT by issuing a DEALLOCATE statement, ECSS II reassigns the idle CRT to the next USER process in line and the process continues about its business. Meanwhile, ECSS II automatically collects queueing statistics for the CRT device so the experimenter will know the length of the waiting line and how long a USER process spent in that waiting line. There are other situations such as this in the model and I will bring them to your attention as we encounter them.

Instead of writing three separate JOBs for each encounter category, one JOB was written with three sections. Entry into a particular section is controlled by a "switch" and a parameter passed to PROG by USER. PROG has three numbered labels: '1', '2', and '3'. Label '1' starts the submodel for the Data Input function. Similarly, label '2' starts the submodel for the Data Base Search and Update function. Label '3', finally, heads the Report Generation section of JOB PROG.

There is a GET command near the beginning of JOB PROG. This simulates the process of obtaining the necessary computer memory into which the program is loaded for execution. If memory enough to satisfy the request is unavailable, ECSS II forms a queue for memory, suspends the JOB, reactivates it when memory does become available, and collects appropriate statistics.

An EXECUTE statement instructs ECSS II to try and obtain the CPU for the program and hold it for the time required to execute the number of instruction indicated. Further on in PROG are EXECUTE statements which have expressions rather than constants as arguments. The ECSS II system immediately starts the simulated execution if the processor is idle. The request, on the

otherhand, is queued and serviced later if the processor is busy. If priorities were used in the model, ECSS II could simulate the preempt-resume method of multiprogramming control. In this model however, the first-come-first-served rule is used. Again, utilization and queuing statistics are kept by ECSS II.

The SEND/RECEIVE statement is associated with simulated IO. It tells ECSS II how much data are to be transferred and over which PATH. Like all the statements encountered thus far, SEND/RECEIVE statements permit queues to form on both devices and paths. The form of SEND/RECEIVE used in the model tells ECSS II to suspend processing, that is, the execution of any statements until the SEND/RECEIVE statement is fully serviced. Thus, delays associated with obtaining the IO devices and the IO path and data transfer time itself are additive to account for the total IO time. Once the IO is complete, the JOB may wait in a queue to once again secure access to the simulated processor.

ECSS II is implemented as a superset of SIMSCRIPT II. That is, by means of a translator, ECSS II statements, intermixed with SIMSCRIPT II statements, are translated to pure SIMSCRIPT II and compiled with a standard SIMSCRIPT II compiler for ultimate execution. This superset-host technique allows the modeler to represent complex computer systems easily and use the facilities of a powerful simulation language. In addition, this implementation allows complex models to be built, debugged, run, and changed with a minimum of effort. ECSS II and languages like it are powerful tools and are cost/effective when applied to problems like the one illustrated here.

There is still one more JOB to look at, the one called LISTOFF. This JOB simulates the spooling process to the PRINTER. The labels '1', '2', and '3' have the same meaning as are used in JOB PROG and provide a means to control the size and amount of data generated per the requirement statements given previously. Here again, the model uses the ALLOCATE statement to control access to the printer, establish a queue, and obtain statistics.

There are several support subroutines needed to start the model, read input variables, and control statistical reports. Although these routines are important, for the sake of brevity they will not be described further.

Experimentation Plan

Experimentation can be an extremely complex process even with a simple simulation model such as the one used for this illustration. So, in keeping with simplicity, some ground rules are needed to narrow the scope of experimentation. First, the parameters which describe the requirement are kept constant.

Table 5 is a complete list of the requirement parameters and their values given in the order of input to the model. In addition, certain design parameters are kept constant. These values are given in Table 6.

PARAMETER	DATA INPUT FUNCTION	DATA BASE SEARCH AND UPDATE	REPORT GENERATION
1. Stimulus Input Character Stream (Characters) (CHAR)	40	20	15
2. Think Time (Seconds) (THINK)	20	20	15
3. MAXIMUM No. of Interactions (REP)	25	10	3
4. Minimum No. of Data Base Accesses (DBACC(1,))	2	50	10
5. Maximum No. of Data Base Accesses (DBACC(2,))	2	100	20
6. Mean Encounter Interarrival Time (Seconds) (IN.ART)	120.	300.	600.

BASELINE REQUIREMENT PARAMETERS

TABLE 5

PARAMETER	VALUE
Length of Simulation Run	3600 Seconds
Data Input Program Core Size (CORE(1))	15000 Bytes
Data Base Search and Update Program Core Size (CORE(2))	60000 Bytes
Report Generation Program Core Size (CORE(3))	30000 Bytes
LISTOFF Program Core Size (LIST.CORE)	10000 Bytes
Data Base Record Size (BLK.SIZE)	2000 Bytes
CRT Screen Capacity (SCREEN)	1920 Bytes

BASELINE CONSTANT DESIGN PARAMETERS

TABLE 6

The second simplifying ground rule states that only hardware design characteristics such as memory capacity, number of CRT's, number and speed of printers, and the CPU execution rate will be varied from one experimental run to the next. These values are input to the simulation through the SYSTEM DESCRIPTION section of the model.

The final simplification involves the performance measures used to evaluate each experimental run. If anything, a simulation provides too much performance data and all too often it is precise to the fifth decimal position while its accuracy may be questionable. The principle performance measure is the average response time for each encounter category. These times are measured in seconds and consist of two components: time spent by encounter material entities in a queue prior to being worked on by a system operator and the time

spent in processing the encounter material including print processing. The total average response time will be reported for each category along with the average total response time across all categories.

There are other performance measures which must not be neglected. These measure performance relative to the utilization and queueing characteristics for the various computer system resources. Accordingly, we will look at measures associated with the CRT's, CPU, PRINTER, PATH, and memory devices.

In all, twelve experimental runs were made with a partial evaluation following each run to determine the set of inputs appropriate for the next run. As stated earlier, experimental strategy is not the principle topic of discussion, so all experimental results will be presented more or less at one time in the discussion which follows.

Experimental Results

To begin with, look at the results from our baseline design. Remember, that our initial timing study predicted an average response time for the three encounter categories of 290.281 seconds, 227.724 seconds, and 338.426 seconds, respectively. With this in mind, Table 7 should reveal some startling results.

CRT = 3
KIPS = 100
MEMORY = 130000
PRINTERS = 1/200 LPM

MEASURE	DATA INPUT FUNCTION	DATA BASE SEARCH AND UPDATE	REPORT GENERATION
QUEUE TIME	1339.1	492.6	352.7
INTERNAL TIME	481.6	1357.0	897.1
TOTAL TIME	1820.7	1849.6	1249.8
INITIAL PREDICTION	290.3	227.7	338.4
AVERAGE RESPONSE TIME		1640.0	
PREDICTED AVERAGE RESPONSE TIME		285.5	

BASELINE DESIGN: SIMULATION RESULTS
RESPONSE TIMES

TABLE 7

Why did we get such results? To answer this question we must look at the detail performance measures given in Table 8. First, waiting time in the encounter material queue (EMQ) is quite high at 1086.8 seconds. Given that the average response time is 1640.0 seconds, the average wait time in the EMQ accounts for over 66 percent of this figure. Second, the table shows that CRT utilization is almost 100 percent and the CPU is busy a little over three-quarters of the sampling interval. It seems likely that the system needs more CRT devices to increase the level of concurrent processing.

Apparently our initial estimate was faulty because it assumed that contention for computer system resources would be minimal. However, there is contention evident. The data shows that the CPU queue (CPUQ) is empty 56.7 percent of the sampling interval and the length of this queue averages 0.7, a small yet significant number. Waiting time for memory averaged 37.5 seconds for each memory request which represents approximately 4 percent of the average internal processing time. A similar analysis using the waiting time to start printing figure of 101.5 seconds yields a percent of average internal time equal to 11.1.

MEASURE	VALUE
CRT UTILIZATION (%)	99.3
ENCOUNTER MATERIAL QUEUE (EMQ) LENGTH	19.0
WAIT TIME IN EMQ	1086.8
CPU UTILIZATION (%)	75.8
CPU QUEUE (CPUQ) LENGTH	.7
PERCENT CPUQ EMPTY	56.7
PRINTER QUEUE (PQ) LENGTH	.564
WAIT TIME IN PQ	101.5
MEMORY UTILIZATION (%)	77.7
MEMORY REQUEST QUEUE (MRQ) LENGTH	.448
WAIT TIME IN MRQ	37.5

BASELINE DESIGN: SIMULATION RESULTS
PERFORMANCE MEASURES

TABLE 8

It is clear that more CRTs are required. Less clear, but nevertheless apparent, is an indication that memory capacity should be increased along with the CPU execution rate, number of printers and printer speed. But how much and in what combinations?

Table 9 gives a complete picture of the experiments run showing the design changes made and the performance measures obtained. Before we examine the numerical values for performance measures in Table 9, look at how the experiments were conducted. The only change made to the baseline to get Experiment 1 was to double the number of CRT's. Next, the memory capacity was doubled for Experiment 2. Experiments 3 and 4 repeat Experiments 1 and 2 but with two printers instead of one. Experiments 5 and 6 repeat Experiments 3 and 4 but with a CPU execution rate of 150,000 instruc-

	EXPERIMENT										
	1	2	3	4	5	6	7	8	9	10	11
NO. CRT'S	6	6	6	6	6	6	6	6	6	6	8
KIPS	100	100	100	100	150	150	150	150	150	150	200
MEMORY SIZE	130	260	130	260	130	260	130	260	130	260	260
NO. PRINTERS	1	1	2	2	2	2	1	1	2	2	2
PRINTER SPEED	200	200	200	200	200	200	600	600	600	600	600
CRT UTIL. (%)	93.3	90.7	90.3	93.8	90.1	93.3	91.3	92.6	93.0	93.8	88.7
EMQ LENGTH	5.2	12.3	8.4	7.5	10.5	5.3	7.7	2.7	6.6	9.1	2.93
EMQ WAIT TIME (SEC)	296.3	707.0	481.7	430.9	603.1	301.9	442.0	155.7	379.6	520.0	167.4
CPU UTIL (%)	77.4	90.0	81.5	85.8	71.1	80.6	67.7	72.8	93.3	70.5	89.1
CPUQ LENGTH	.73	2.28	.96	2.57	.51	2.36	.54	1.69	.60	1.87	4.09
CPUQ EMPTY (%)	53.4	28.4	47.3	27.2	63.4	35.3	64.1	46.5	62.0	47.1	19.1
PQ LENGTH	3.16	3.40	.18	.96	.08	1.25	.21	1.49	.00	.18	1.50
PQ WAIT TIME (SEC)	284.3	408.2	16.9	86.2	7.5	88.3	17.4	105.8	.09	17.6	110.5
MEMORY UTIL. (%)	87.4	65.4	88.5	69.4	89.5	69.6	87.1	73.0	86.4	60.2	80.4
MRQ LENGTH	1.96	0.0	2.36	0.0	2.07	.09	1.63	.09	2.24	.11	.27
MRQ WAIT TIME (SEC)	89.1	0.0	106.2	0.0	91.1	3.2	63.2	3.1	94.0	4.8	9.7
AVERAGE RESPONSE TIME	1039.6	1581.6	1115.0	1024.8	1184.4	923.7	939.2	655.5	927.0	1017.1	889.0

EMQ - Encounter Material Queue
CPUQ - CPU Queue
PQ - Printer Queue
MRQ - Memory Request Queue

EXPERIMENTAL RUNS: SIMULATION RESULTS - PERFORMANCE MEASURES

TABLE 9

tions per second instead of 100,000. Experiments 7, 8, 9, and 10 all have a CPU execution rate of 150,000 instructions per second, a 600 line per minute printer, but are otherwise repeats of Experiments 1, 2, 3, and 4. This design explores that portion of the performance response surface which can reasonably be considered state-of-the-art and responsive to the requirement given earlier. Experiment 11 is a run to see the results from a design possessing abundant system resources.

Now, what do these data tell us about the proposed design alternatives? First, the differences between Experiments 1 and 2 are interesting. Memory capacity doubles, yet average response time increases with all other design parameters held constant. The reason is quite straight forward. With more jobs in memory, the CPUQ is longer and, on the average, the CPUQ is less frequently empty. CPU contention is much greater in Experiment 2 because more jobs can be in memory and issue a greater number of requests for the CPU resource. In a like fashion, more jobs processed means more print output produced in a shorter time. This is indicated by an almost two-fold increase in PQ wait time. The average memory queue length is zero which indicates that we probably didn't need 260,000 bytes of memory in the first place. The reader should examine the data closely to see if other such anomalies can be found and explained.

The second basic observation is that Experiment 8 yields the best average response time of 655.5 seconds. This is partially explained by the low EMQ wait time of 155.7 seconds. Further explanation is provided by observing that CPU utilization, relatively speaking, is low at 72.8 percent and memory utilization is similarly low at 73.0 percent. Taken together, this indicates that overall system resource demands are balanced better with this configuration than with the others. That is, on a relative basis, CPU contention is lower, allowing individual jobs to proceed to an IO operation and thereby release the CPU resource for other jobs. Progress in using the CPU resource is aided considerably by the 150,000 instruction per second execution rate.

Finally, there is a major observation that must be made. By the very nature of the experiments, it is impossible to know what performance gains can be achieved realistically if design changes are made to the simulated software. More will be said on this point later. But for now, there is a performance limit beyond which we cannot go given the nature of the experiments used thus far in the illustration.

By way of summary, it was shown that the simulation model could be used to evaluate the baseline configuration proposed earlier. In addition, a series of experiments can be run each using a slightly altered set of input design parameters. Finally, from examination of the performance output measures, it is possible to select one or more design alternatives that exhibit desirable performance characteristics.

RESTATEMENT

In the previous sections, five out of the six steps in the Methodology were explained and illustrated. In the interest of brevity, the documentation requirements were not stressed. One illustration, however, does not conclusively demonstrate the usefulness of any procedure nor does it adequately warn potential user of the pitfalls usually encountered during design evaluation activity.

The model developed for the illustration can be easily extended to examine many possible software design implementations in addition to the hardware designs shown in this paper. New models appropriate for different application areas can also be easily written using ECSS II. Some examples of software related implementations which might be addressed by an ECSS II model are listed below:

- Prempt-resume CPU dispatching algorithms.
- Priority driven Job scheduling.
- Priority driven IO scheduling.
- Program segmentation and paging.
- Program overlay structures.
- Working set page allocation algorithms.
- Double buffering of IO.
- Multiprocessor task dispatching.
- Dual channel disk access control algorithms.

Basically, this paper presented two propositions. First, that computer system design evaluation activities should follow the six steps outlined and be conducted by a design evaluation team made up of users, design specialists, and development specialists. Early use of the procedure introduces discipline to the process which might otherwise be driven by emotional bias. Use of an integrated teams of users, designers, and developers, tends to minimize the communication gap problems and hopefully shortens the overall process.

The second proposition stated that computer system simulation is a valuable tool which can be effectively used during the validation phase to articulate designs in the form of readable models and to gather valuable estimates of performance for each proposed design. In an important sense, computer system simulation studies are to the evaluation of computer system designs what wind tunnel tests are to the evaluation of air vehicle designs. For years, Program Managers responsible for aeronautical system developments have relied on the results from wind tunnel tests. The same opportunities for evaluating and reducing the development risk associated with embedded computer systems are now available to Program Managers through the use of computer system simulation.

In concluding, it must be stated that no technique, model, or procedure can totally eliminate the risks associated with developing complex weapon systems containing embedded computers. The procedures outlined and illustrated in this paper are proposed because they tend to provide information about and opportunities to evaluate computer system designs in a logical way and in an environment where hypotheses can be tested. As such, computer system simulation and the six step procedure in which it is employed can significantly assist in reducing design risk, but it will never totally eliminate it.

BIBLIOGRAPHY

1. Bell, T. E., Boehm, B. W., and Watson, R. A., "Computer System Performance Improvement: Framework and Initial Phases," RAND Corp., R-549-PR, August 1971.

2. Bell, T. E., "Computer Performance Analysis: Objective and Problems in Simulating Computers," RAND Corp., R-1051-PR, July 1972.

3. Kiviat, P. J., "Digital Computer Simulation: Computer Programming Languages," RAND Corp., RM-5883-PR, January 1969.

4. Kiviat, P. J., Villaneuva, R., Markowitz, H. M., "SIMSCRIPT II.5 Programming Language," Consolidated Analysis Centers, Inc., Los Angeles, CA, March 1973.

5. Kosy, D. W., "Experience with ECSS," RAND Corp., R-560-NASA/PR, December 1970.

6. Kosy, D. W., "The ECSS II Language for Simulating Computer Systems," RAND Corp., R-1895-GSA, December 1975.

7. Nielsen, N. R., "ECSS: An Extendable Computer System Simulator," RAND Corp., RM-6132-NASA, February 1970.

8. Watson, R., "Computer Performance Analysis: Applications of Accounting Data," RAND Corp., R-573-NASA/PR, May 1971.

APPENDIX A

STAT.NOTE - Temporary entity used to represent the encounter material and to retain statistical values collected as the entity passes through the simulated system.

ST.ATIME - Attribute which saves the arrival time of a STAT.NOTE.

ST.STIME - Attributes which saves the starting time for processing the encounter material on the computer system. NOTE: ST.STIME-ST.ATIME is the time spent waiting to begin processing.

ST.RTIME - Attribute which saves the time that the final response is provided for the encounter.
NOTE: ST.RTIME-ST.STIME is the time spent processing the encounter on the computer system.
NOTE: ST.RTIME-ST.ATIME is the total response time for the encounter material.

ST.TYPE - Attribute used to tell JOB PROG what type of encounter is being processed.

1 = Data Input
2 = Data Base Search and Update
3 = Report Generation

ST.EXEC - Attribute which accumulates all CPU execution time for the encounter.

ST.BLOCK - Attribute which accumulates all intervals of time waiting for a resource to be provided (except the CPU).

ST.READY - Attribute which accumulates the time intervals spent waiting for the CPU to be made available.

NOTE.SET - A First-In-First-Out Set used to hold the STAT.NOTEs after they are created.

CHAR - Three element array used to hold the number of input stream characters per encounter input interaction.

CORE - Three element array used to hold the memory bytes required for each encounter process.

REP - Three element array used to hold the upper limit for the number of interactions per encounter.

THINK - Three element array which holds the non-overlapped think time for each encounter interaction.

INT.ART - Three element array which holds the encounter material mean interarrival time used to generate simulated encounter transactions.

ST.RPT - Variable used to input the stop time for the simulation.

DBACC - Two dimensional array of two rows and three columns used to hold the upper and lower data base reference counts for each encounter category.

BLK.SIZE - Variable for the disk data record size in bytes.

LST.CORE - Variable for the size of the LISTOFF program in bytes.

SCREEN - Variable for the size of the simulated CRT screen in bytes.

APPENDIX B

```
/*SIGNON       RM031 A
PREAMBLE
  TEMPORARY ENTITIES...
     EVERY STAT.NOTE HAS
        AN ST.ATIME,
        AN ST.STIME,
        AN ST.RTIME,
        AN ST.TYPE,
        AN ST.EXEC,
        AN ST.BLOCK,
        AN ST.READY
        AND BELONGS TO A NOTE.SET

     DEFINE ST.STIME,ST.RTIME,ST.EXEC,ST.BLOCK,ST.READY,ST.ATIME
                                        AS REAL VARIABLES

  THE SYSTEM OWNS A NOTE.SET
  DEFINE NOTE.SET AS A FIFO SET

  DEFINE CHAR,CORE,REP AS 1-DIMENSIONAL, INTEGER ARRAYS
  DEFINE THINK,INT.ART  AS 1-DIMENSIONAL, REAL ARRAYS
  DEFINE ST.RPT AS A REAL VARIABLE
  DEFINE DBACC AS A 2-DIMENSIONAL, INTEGER ARRAY
  DEFINE BLK.SIZE, LST.CORE,MAX.CNT,SCREEN AS INTEGER VARIABLES
  DEFINE MAX.TIME,ART AS REAL VARIABLES
END      ''PREAMBLE
DEFINITION DESCRIPTION
```

```
  DEFINE UNITS
     1 SECOND = 1 TIME.UNIT,
     1 SECONDS = 1 SECOND,
     .001 SECONDS = 1 MILLISECOND,
     1 MILLISECOND = 1 MILLISECONDS,
     1 MS = 1 MILLISECONDS

  DEFINE UNITS
     1 BYTE = 1 DATA.UNIT,
     1 BYTES = 1 BYTE,
     1000 BYTES = 1 KBYTE,
     1 KBYTE = 1 KBYTES

END      ''DEFINITION DESCRIPTION

SYSTEM DESCRIPTION

  SPECIFY 1 CPU WHICH
     STORES UP TO 2 BATCH JOBS,
     EXECUTES 150000 INSTRUCTIONS PER SECOND,
     HAS CAPACITY OF 260 KBYTES

  SPECIFY 1 CH1 WHICH
     TRANSFERS 3 MESSAGES CONCURRENTLY

  SPECIFY 1 CH2 LIKE CH1 EXCEPT IT
     TRANSFERS 1 MESSAGE

  SPECIFY 8 PRIVATE CRT, EACH
     TRANSMITS 300 BYTES PER SECOND,
     ABSORBS 250.0 MILLISECONDS PER MESSAGE

  SPECIFY 3 DISK, EACH
     TRANSMITS 32000 BYTES PER SECOND,
     ABSORBS  40.0 MILLISECONDS PER MESSAGE

  IDENTIFY DISK#1 AS MASTER.DISK

  SPECIFY 1 PRIVATE ACCTG.FILE

  SPECIFY 2 PRIVATE PRINTER WHICH
     TRANSFERS 1 MESSAGE,
     TRANSMITS 1200 BYTES PER SECOND

  PATH C.PATH CONNECTS CH1 TO CRT
  PATH D.PATH CONNECTS CH2 TO DISK
  PATH P.PATH CONNECTS CH1 TO PRINTER

END      ''SYSTEM DESCRIPTION
WORKLOAD DESCRIPTION

EXTERNAL PROCESS USER GIVEN TYP,ATIME

  DEFINE TYP,PR,C,VOLUME,REPITITIONS,P,I,SN
     AS INTEGER VARIABLES
  DEFINE ATIME
     AS A REAL VARIABLE

  LET .IDENTITY(.JOB) = "USER 0" + TYP
  CREATE A STAT.NOTE CALLED SN                    ''KEEP STATISTICS
  LET ST.ATIME(SN) = ATIME
  LET ST.TYPE(SN) =TYP                            ''SAVE TRANSACTION TYPE
  FILE SN IN NOTE.SET
  ALLOCATE FROM CRT AS C                          ''GET A CRT OR GET IN QUEUE
  LET ST.STIME(SN) =TIME.V                        ''SAVE START TIME
  LET VOLUME = RANDI.F(1,.N.MEMBERS(DISK),1)      ''GEN DISK PACK TO BE USED
  LET REPITITIONS = RANDI.F(2,REP(TYP),1)         ''GEN ITERATION FOR USER
```

```
  LET PR = .PROCESS
  START PROG CALLED P GIVEN PR,VOLUME,
     REPITITIONS,TYP,SN
     ON CPU                                          ''START CPU PROGRAM
  WAIT FOR SIGNAL                                    ''SUSPEND UNTIL PROG IS READY
  FOR I = 1 TO REPITITIONS,                          ''SET UP INTERACTION
     DO...
     SEND CHAR(TYP) BYTES FROM .DEVICE(C) TO CH1 VIA C.PATH TO P
        WAITING FOR A RESPONSE                       ''SUSPEND FOR PROG RESPONSE

     WAIT FOR THINK(TYP) SECONDS                     ''THINK TIME FROM RESPONSE
                                                     ''TO NEXT PROMPT
  LOOP  ''I

  SUBMIT LISTOFF GIVEN TYP,VOLUME,REPITITIONS,SN
     TO CPU                                          ''BATCH JOB TO LIST ANY OUTPUT
  DEALLOCATE THE C                                   ''PHYSICAL RELEASE OF CRT

  TERMINATE

END       ''USER
JOB PROG GIVEN USR,VOL,INTERACTIONS,TYP,SN

  DEFINE USR,VOL,INTERACTIONS,I,J,SN,SPACE
     AS INTEGER VARIABLES

  LET .IDENTITY(.JOB) = "PROG 0" + TYP
  GET CORE(TYP)  BYTES FROM CPU CALLED SPACE         ''GET REQUIRED MEMORY
  ALLOCATE ACCTG.FILE                                ''LOCK ON ACCTG FILE LOGON
  RECEIVE 800 BYTES FROM MASTER.DISK VIA D.PATH WITH PRIORITY 1
  DEALLOCATE ACCTG.FILE                              ''RELEASE ACCTG FILE LOCK ON
  EXECUTE 3500 INSTRUCTIONS                          ''LOGON PROCEDURE
  SIGNAL USR                                         ''TELL USER TO PROCEED

  WHILE INTERACTIONS GT 0                            ''SET UP INTERACTIONS LOOP
     DO...
     WAIT FOR INPUT                                  ''SUSPEND FOR USER PROMPT
     GO TO 1,2,3 PER TYP                             ''GOTO SUBPROCESS PER TR TYPE

  '1'          ''INPUT DATA FUNCTION
     EXECUTE 2500+CHAR(TYP) INSTRUCTIONS             ''INTERPRET INPUT FROM USER
     SEND CHAR(TYP) + 200 BYTES TO DISK#VOL VIA D.PATH
                                                     ''PUT INPUT DATA ON DISK
     EXECUTE 4000 INSTRUCTIONS                       ''UPDATE INDEX FORMAT RESPONSE
     SEND CHAR(TYP) BYTES VIA C.PATH AS A RESPONSE   ''SEND RESPONSE TO USER
     GO TO CONTINUE

  '2'          ''SEARCH-UPDATE FUNCTION
     EXECUTE 500 *CHAR(TYP)  INSTRUCTIONS            ''SCAN INPUT FOR REQUEST
     LET J = RANDI.F(DBACC(1,TYP),DBACC(2,TYP),1)    ''GEN ACCESSES TO FIND DATA
     FOR I = 1 TO J
        DO...
        RECEIVE BLK.SIZE BYTES FROM DISK#VOL VIA D.PATH  ''READ DISK BLOCK
        EXECUTE 50*BLK.SIZE INSTRUCTIONS             ''EXAMINE BLOCK FOR REQ DATA
     LOOP  ''I
     EXECUTE   200 *J INSTRUCTIONS                   ''PROCESS DATA RETRIEVED
     SEND BLK.SIZE BYTES TO DISK#VOL VIA D.PATH
                                                     ''UPDATE DISK BLOCK PER REQUEST
     SEND SCREEN BYTES VIA C.PATH AS A RESPONSE      ''SEND RESPONSE TO USER
     GO TO CONTINUE

  '3'        ''REPORT GENERATION FUNCTION
     EXECUTE 500 * CHAR(TYP)  INSTRUCTIONS           ''INTERPRET REQUEST
     LET J = RANDI.F(DBACC(1,TYP),DBACC(2,TYP),1)
     FOR I =1 TO J
        DO...
        RECEIVE BLK.SIZE BYTES FROM DISK#VOL VIA D.PATH   ''GET REPORT DATA
        EXECUTE 15 *BLK.SIZE + 100 INSTRUCTIONS  ''FORMAT HEADING AND REPORT
        SEND (BLK.SIZE+0.20*BLK.SIZE) BYTES TO DISK#VOL
          VIA D.PATH                                 ''FORMATTED OUTPUT TO DISK SPOL
```

```
      LOOP  ''I
      EXECUTE 4000 INSTRUCTIONS                    ''PREPARE USER RESPONSE SUMMRY
      SEND CHAR(TYP) BYTES VIA C.PATH
         AS A RESPONSE                             ''SEND RESPONSE TO USER
    'CONTINUE'

      SUBTRACT 1 FROM INTERACTIONS

    LOOP  ''INTERACTIONS

    EXECUTE 2500 INSTRUCTIONS                      ''CLEANUP FOR LOGOFF
    ALLOCATE ACCTG.FILE                            ''LOCK ON ACCTG FILE FOR LOGOFF
    RECEIVE 800 BYTES FROM MASTER.DISK VIA D.PATH  ''GET ACTG BLOCK FOR UPDATE
    SEND 100 BYTES TO MASTER.DISK VIA D.PATH       ''UPDATE ACCTG FILE
    DEALLOCATE ACCTG.FILE                          ''RELEASE ACCTG FILE
    ADD .EXECUTION.TIME(.JOB) TO ST.EXEC(SN)
    ADD .BLOCKED.TIME(.JOB) TO ST.BLOCK(SN)
    ADD .READY.TIME(.JOB) TO ST.READY(SN)
    FREE  THE SPACE                                ''ALL DONE FREE CORE SPACE

    TERMINATE

  END       ''PROG
  JOB LISTOFF GIVEN T,V,BLK,SN

    DEFINE T,V,BLK,I,PRT,SN,SPACE,J,SIZE
           AS INTEGER VARIABLES

    LET .IDENTITY(.JOB) = "LSTF 0" + T
    GET LST.CORE BYTES FROM CPU CALLED SPACE       ''OBTAIN NEEDED CORE FOR JOB
    ALLOCATE PRINTER AS PRT                        ''OBTAIN PRINTER FOR OUTPUT
    EXECUTE 5000 INSTRUCTIONS                      ''INITIALIZATION
    GO TO 1,2,3 PER T
  '1'
    LET SIZE = CHAR(T)    + 200
    GO TO CONTINUE
  '2'
    LET SIZE = BLK.SIZE + 0.20 * BLK.SIZE
    GO TO CONTINUE
  '3'
    LET J = 0
    FOR I = 1 TO BLK
       LET J = J + RANDI.F(DBACC(1,T),DBACC(2,T),1)
    LET BLK = J
    GO TO 2
  'CONTINUE'
    FOR I =1 TO BLK                                ''SETUP OUTPUT LOOP BY BLOCK
       DO...
                                                   ''READIN DATA BLOCK FOR REPORT
       RECEIVE SIZE BYTES FROM DISK#V VIA D.PATH
                                                   ''SEND OUTPUT TO SELECTED PTR
       EXECUTE 5*SIZE INSTRUCTIONS
       SEND SIZE BYTES TO .DEVICE(PRT) VIA P.PATH
    LOOP  ''I
    LET ST.RTIME(SN) = TIME.V                      ''SAVE FINISH TIME FOR TRANSAC
    ADD .EXECUTION.TIME(.JOB) TO ST.EXEC(SN)
    ADD .BLOCKED.TIME(.JOB) TO ST.BLOCK(SN)
    ADD .READY.TIME(.JOB) TO ST.READY(SN)
    FREE THE SPACE                                 ''FINISHED FREE MEMORY USED
    DEALLOCATE THE PRT
    TERMINATE

  END       ''LISTOFF
  END       ''WORKLOAD DESCRIPTION
```