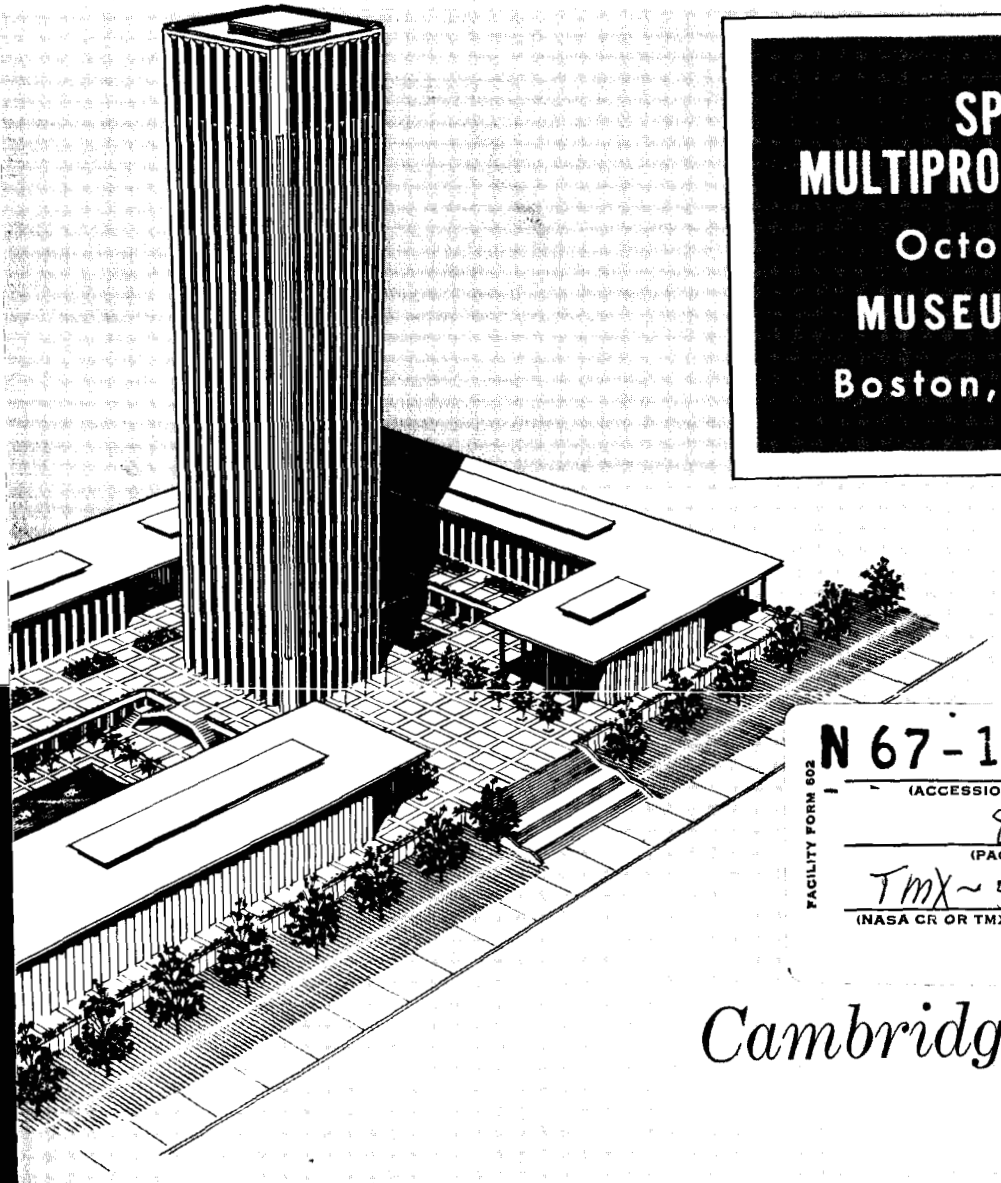
11

ELECTRONICS RESEARCH CENTER



SPACEBORNE MULTIPROCESSING SEMINAR

October 31, 1966

MUSEUM OF SCIENCE

Boston, Massachusetts

Cambridge, Massachusetts

SPACEBORNE MULTIPROCESSING SEMINAR

MUSEUM OF SCIENCE
Boston, Massachusetts

October 31, 1966

Sponsored by

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
ELECTRONICS RESEARCH CENTER
Cambridge, Massachusetts

FOREWORD

As the Nation's space program evolves, increased emphasis is being placed on developing the computer technology to support an expanding class of on-board computational tasks. Within this framework, various multiprocessing approaches appear to offer strong promise of accommodating such complex requirements in future missions.

We have been most fortunate in arranging an outstanding group of speakers, each of whom was selected on the basis of important contributions to computer technology. Capsule summaries of the presentations have been compiled in this document so that all attendees may have a background reference for the papers to be delivered.

I am especially grateful to all those present, both for their interest in the proceedings and for taking time out from their busy schedules to participate in the Seminar.

Thomas E Burke

THOMAS E. BURKE
General Chairman

CONTENTS

REQUIREMENTS / ORGANIZATION

MULTIPROCESSOR ORGANIZATION FOR MANNED MARS MISSION

A. O. WILLIMAN

Mr. Williman is Chief, Advanced Digital Systems Group, Data Systems Division, Autonetics, a division of North American Aviation. Mr. Williman has been with Autonetics since 1959. Since then he has worked on Minuteman Systems Engineering, determination of computer requirements for future applications, and advanced systems analysis. Prior to joining Autonetics, he worked in the Missile Division of North American Aviation on advanced missile design projects and at Hallamore Electronics Co., on the F106 aircraft projects. He obtained his M. S. M. E. degree in 1957 from the University of Southern California and his B. S. E. degree in 1952 from U. C. L. A.

L. J. KOCZELA

Mr. Koczela is a Senior Research Engineer, Advanced Digital Systems Group, Data Systems Division, Autonetics, a division of North American Aviation. Mr. Koczela has been with Autonetics since 1963, primarily engaged in determining computer requirements and computer systems definition for numerous spaceborne projects. Currently he is assigned as Principal Investigator of the NASA Spaceborne Multiprocessing Study. Prior to joining Autonetics, he worked at the Missile and Space Division of General Electric on electronic systems for space applications. He obtained the M. S. E. E. degree in 1963 from the University of Pennslyvania and the B. S. E. E. degree in 1961 from Rutgers University.

GERALD J. BURNETT

Mr. Burnett is a Senior Research Engineer, Advanced Digital Systems Group, Data Systems Division, Autonetics, a division of North American Aviation. Mr. Burnett has been with Autonetics since September 1965, working primarily with multiprocessing organizations and silicon sapphire technology. He is currently investigating computer designs for a NASA Spaceborne Multiprocessing Study. Prior to joining Autonetics, Mr. Burnett worked at Project MAC, M. I. T., while obtaining his M. S. E. E. degree from M. I. T. (1965). He received his B. S. E. E. degree from M. I. T. in 1964.

MULTIPROCESSOR ORGANIZATION FOR MANNED MARS MISSION

By A. Williman, L. Koczela, G. Burnett

Autonetics, A Division of North American Aviation, Inc., Anaheim, California

SUMMARY

Three different approaches to computer organization for a long duration manned space mission are given. Flexibility in meeting computational requirements is an important factor in the computer system design. The organizations are currently being evaluated, and preliminary evaluations indicate that a Multiprocessor or Distributed Logic approach offers the most potential for future Space missions.

INTRODUCTION

The purpose of this paper is to review some of the considerations in the application of digital computers to long duration space missions. The space missions to be considered are the extended manned missions which are typical of advanced earth orbital and manned planetary missions. A manned Mars lander mission would typically have a duration of 420 days, an earth orbiting space station might have a mission duration of 1 year; to develop rather specific requirements for computer design, a manned Mars mission was considered in detail. Although some of the processing tasks are unique to this mission, the resulting computer configurations will in general be applicable to other space missions.

REQUIREMENTS

The desirability for some form of digital computation aboard the vehicle has been demonstrated in the on-board control of Gemini reentry and will be further shown in Apollo. The Mars mission computational tasks which can be mechanized on a digital computer system are shown in Figure 1. They are basically divided into two groups: Command and Control and Mission Data Processing. The computational requirements vary considerably from phase to phase during a mission, as shown in Figures 2 and 3. The reliability requirements for the Mars mission were defined as a 0.997 probability of success and a 0.997 availability for approximately 10,000 hours.

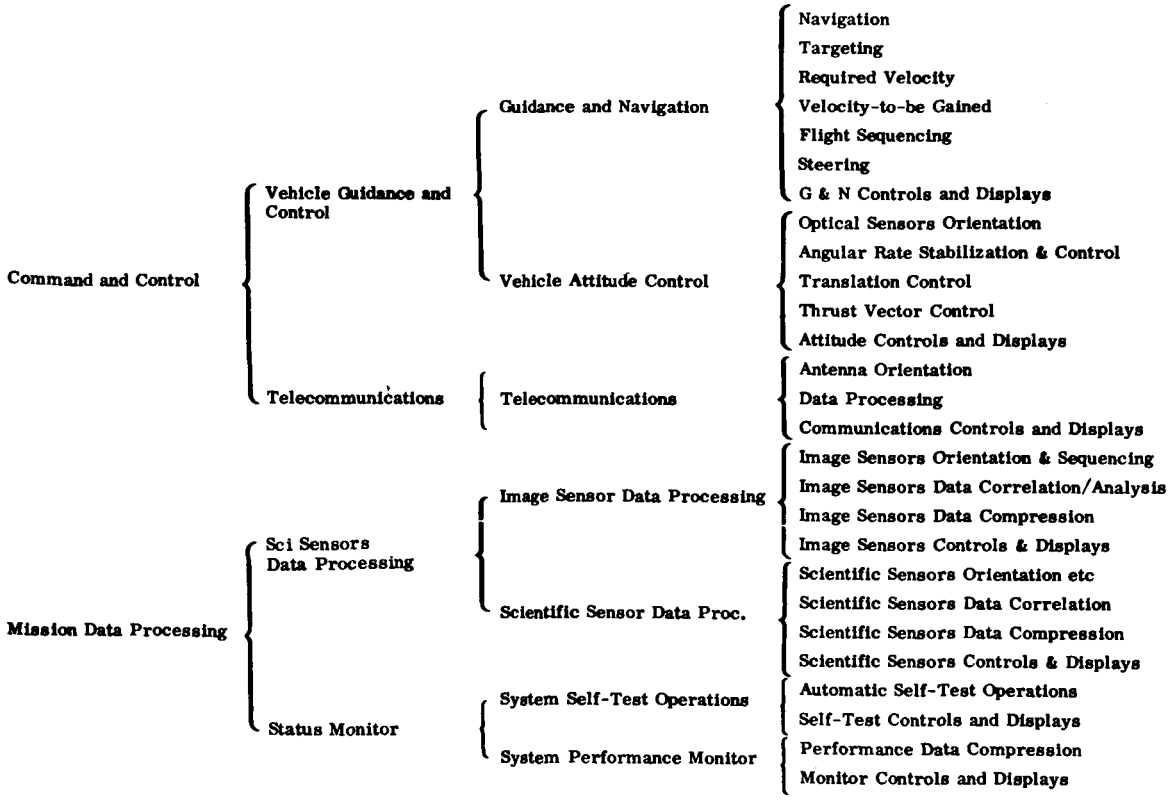


Figure 1. Computational and data processing functions

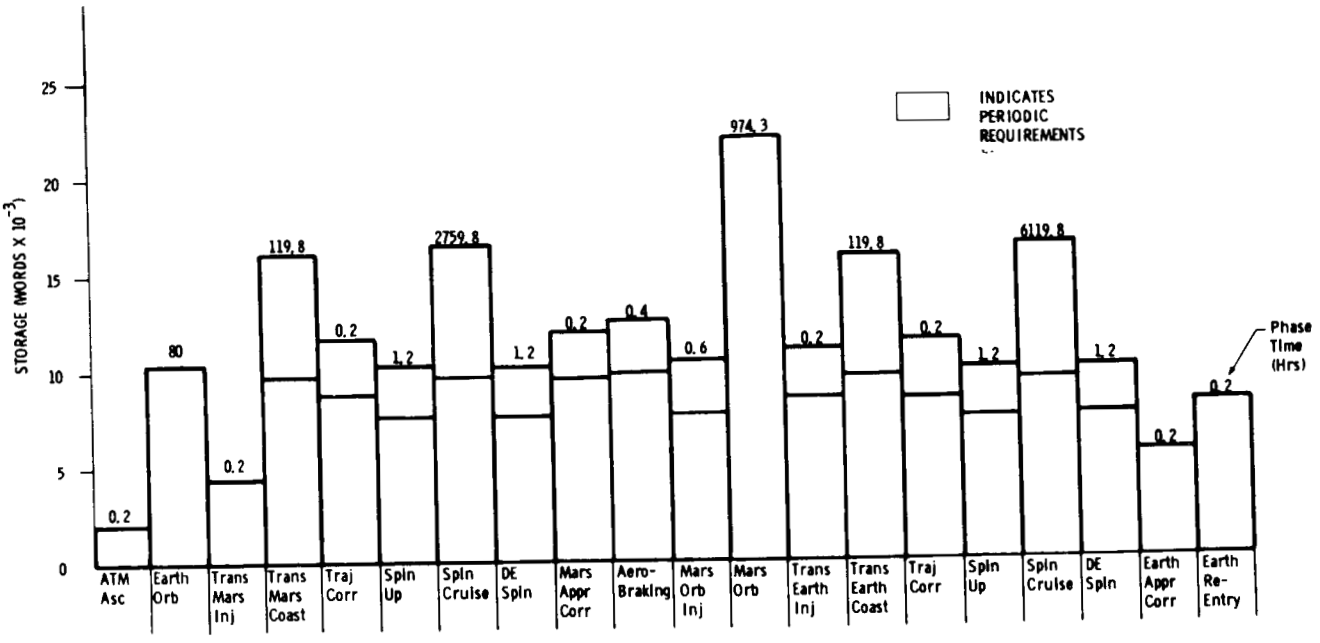


Figure 2. Memory requirements per manned Mars mission phase

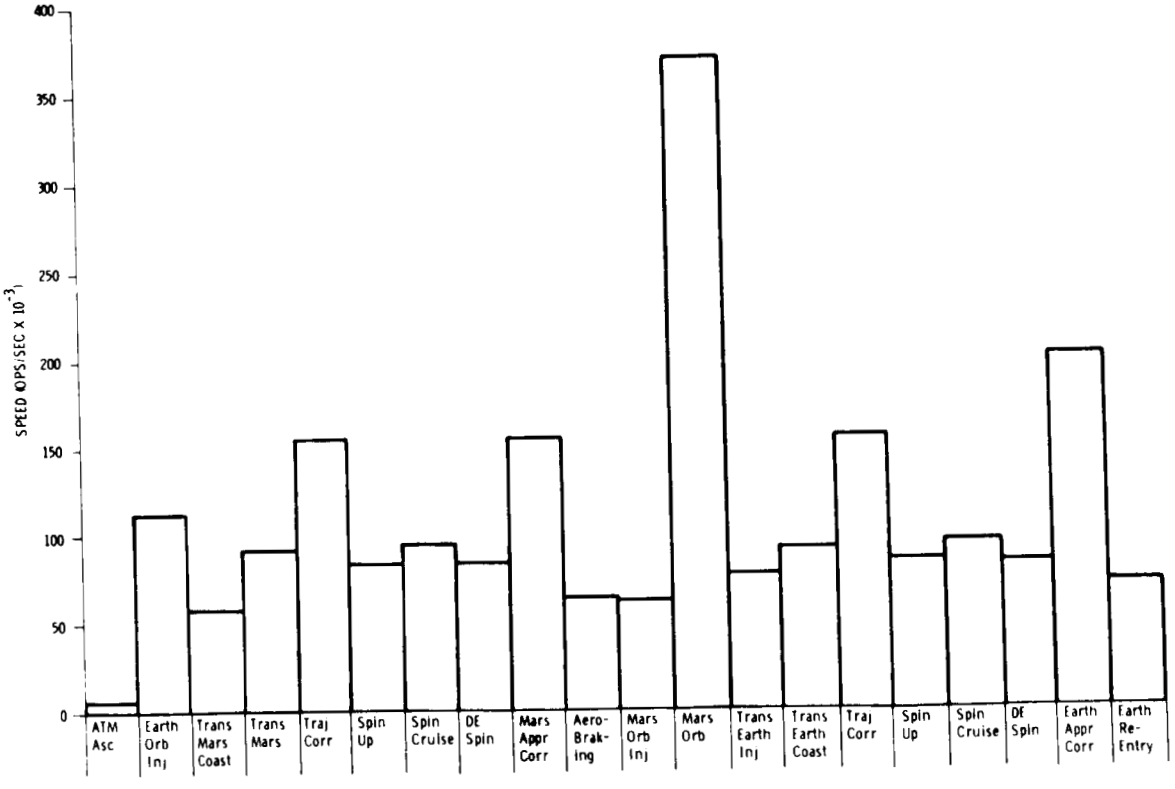


Figure 3. Computational speed requirements per manned Mars mission phase

MODULAR COMPUTATION SYSTEMS

A single computer is quite efficient for a specific requirement, however, as soon as that requirement is altered, such as the speed requirement doubled, difficulty is encountered. For this reason and others as pointed out below the potential of a modular computational approach was investigated. What are the advantages expected from varying the computational capability of the computer system? Power (which is very important for long duration missions) can be saved by being able to turn modules on and off. Reliability is increased, given that failure rates of dormant equipment are lower than operating equipment (which appears to be the case from preliminary data). Probability of mission success and computer availability are greatly enhanced due to the capability of withstanding failures by reconfiguration at the module level. Only a portion of the computational capability may be lost during a failure, thereby giving the possibility of "graceful degradation."

The computational requirements have been shown to vary considerably during a typical long duration space mission. If the computer system were designed in terms of modules, the potential of turning off some of the modules during various phases would exist. As an example in the Mars mission considered, if two computer modules are designed to share the load during the Mars orbital phase, then only one of these modules may be required during the long duration coast and cruise phases; this capability results in advantages as given above, most notably in power and reliability.

In addition to the above, the computational requirements are expected to vary substantially from mission to mission (for example the manned Mars Lander vs the unmanned Mars flyby) and in fact, from the Mission Module of a spacecraft to the Lander Module. A modular concept then enables the setting up of a baseline set of computing modules upon which many computer systems can be built. This enables a common sparing philosophy on any particular mission as well as common sparing, development, production, and testing for a range of missions.

Another point worthwhile mentioning is the use of the modules to back up each other during critical mission phases such as Mars Orbit Injection and Earth Reentry. During these phases, 5 seconds is typical of the maximum allowable time for switching to a backup or redundant system in case of a failure. This requires fault detection, isolation, and reconfiguration to an on-line backup within 5 seconds. The critical computations occur in nonheavily loaded phases and as a result the extra modules necessary only in the heavily loaded Mars Orbital phase can be used as on-line backup; however, even if the extra modules were not available, modules not carrying out critical computations could be used as backups in order to be able to withstand multiple failures. This shows that all the modules in the system can be used in order to obtain a very high probability of carrying out critical computations (probability of success).

The discussion so far has indicated some advantages if so called modularity is introduced into the computational system. The question which now must be answered is how to attain this modularity. It is then possible to estimate what the modularity costs so that an evaluation can be made to determine the most effective approach. Three computer system organizations have been investigated to attain modularity. These are Multiple Computer, Multiprocessor and Distributed Logic Organizations.

Multiple Computer

The Multiple Computer system consists of two independent computers each with a 200,000 operation per second processor, a 24K - 18 bit word memory, and a program controlled I/O section. The individual computers perform self-checks and are interconnected so that critical system outputs can be automatically switched from a failed computer to a correctly operating computer through an output switch. A block diagram of the system is shown in Figure 4. The processor features shown in this figure were developed from an analysis of the computational requirements. The software considerations for this organization and the other two organizations are given in a following paper by R. Hokom[1] entitled "Executive Program Control for Spaceborne Multiprocessors."

Multiprocessor

The Multiprocessor, shown in Figure 5, consists of two 200,000 operation per second processors, three 12K - 18 bit word memories and two program-controlled I/O units with full intercommunication between the processors and the other modules in the system. The requirements for this system are the same as those for the Multiple Computer; as a result the processor features are the same. It should be noted however, that only three 12K memory modules were necessary instead of the four in the Multiple Computer. This more efficient use of the memory was obtained due to the flexible processor-memory communication; however, there will be approximately 2.5 times as many lines required for the Multiprocessor than for the Multiple Computer. The memories and the input/output units will have lock-out features which will permit the multiprocessor to operate essentially as a Multiple Computer system during critical mission phases where redundant calculations are required. This prevents the failure of one processor-memory system from annihilating information in memories or in disturbing the operation of the other processing system.

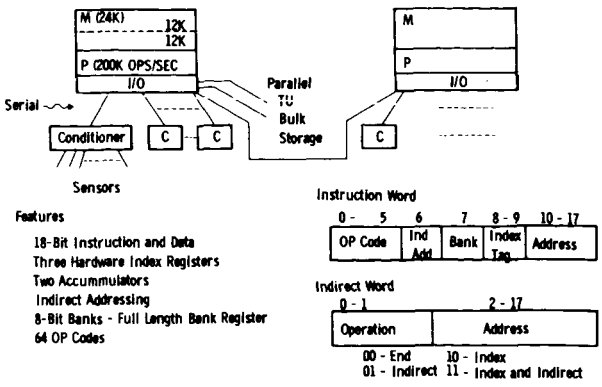


Figure 4. Multiple computer organization

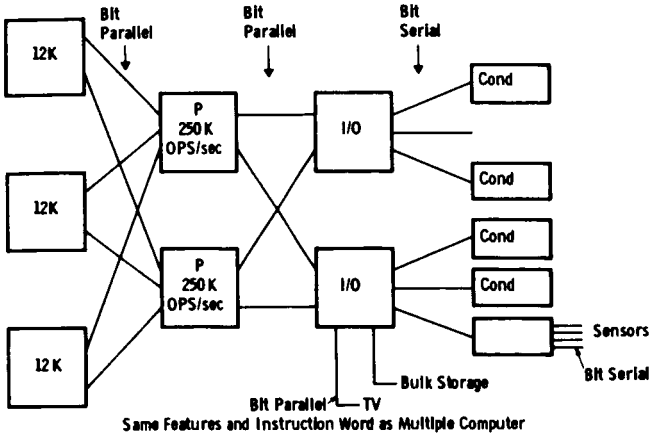


Figure 5. Multiprocessor organization

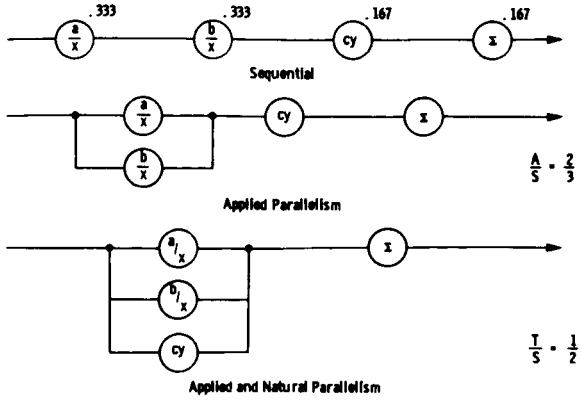


Figure 6. Example of parallelism

Distributed Processor

The third type of computer organization considered is a distributed logic organization. This type of computer contains a decentralization of the logic elements on an array basis. Each element or cell communicates with a number of other cells, and each cell has some memory associated with it. The complexity of a cell can vary from the execution of a single instruction to a small computer. The control of the execution of a program in the array can either be distributed among the cells or can be centrally controlled. The array type construction as depicted in the Holland machine[2] is typical of a local control type configuration. The Solomon computer[3] is typical of the global control type configuration. Determination of which approach to follow was one of the first requirements of the distributed logic study. To aid in this determination it was necessary to define the parallelism associated with computational problems. Two types of parallelism have been defined: natural parallelism which has the property for carrying out a number of operations on distinct data bases or on the same data base simultaneously and independently; the second, Applied parallelism where a number of exactly the same operations on distinct data bases or on the same data base are carried out simultaneously. Figure 6 is an example of applied and natural parallelism for a sum of products type computation. The sequential operation is shown at the top of the figure. The applied parallelism example shows the identical operations a/x and b/z being computed simultaneously. The computation c x y is done sequentially and the sum is made sequentially. At the bottom of the figure the use of natural parallelism is shown by the parallel solution of c x y. The saving in time by the use of parallelism is shown at the right hand side of the figure with a solution time for applied parallelism being 2/3 of that for the sequential computation and with the applied and naturalism being 1/2 that of the sequential solution time. Computational requirements for the Mars mission were examined to determine the amount of applied parallelism and natural parallelism that could be mechanized. Figure 7 shows the improvement in computational speed as a function of the number of cells in applied parallelism. It can be seen from the curve that after 25 cells, there is little reduction in the

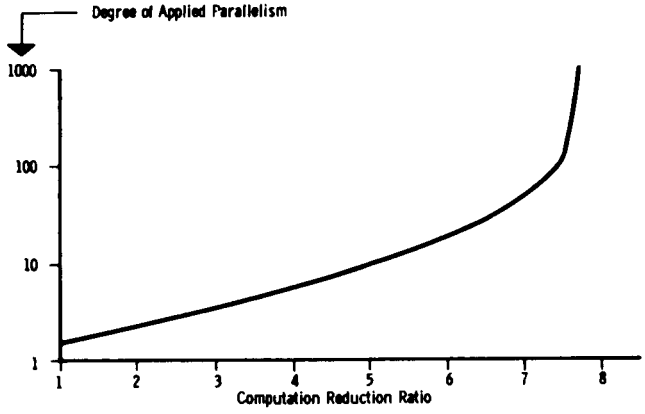


Figure 7. Applied parallelism degree of complexity vs computation speed

computational speed. Figure 8 shows the effect of natural parallelism on the computational task speed reduction; this figure shows that after six computational centers the reduction in computational speed is negligible.

The chosen Distributed Processor was developed to take advantage of both natural and applied parallelism and also to take maximum advantage of the technology assumed for 1980. The organization of the Distributed Processor is shown in Figure 9 in block diagram form. It consists of a number of groups of cells all interconnected through an inter-group bus. Each cell executes macro instructions from storage or from a controller cell and can communicate to its neighboring cells. It is seen that the cells are organized into fixed sized groups each of which can perform a computational task. This alleviates the optimization problems for program reconfiguration and also makes executive monitoring simpler. The number of cells in the group is chosen to best meet the applied and natural parallelism inherent in the Mars Lander Mission computations. Therefore a structure containing approximately 25 cells per group and having up to 25 groups was specified. Each cell within a

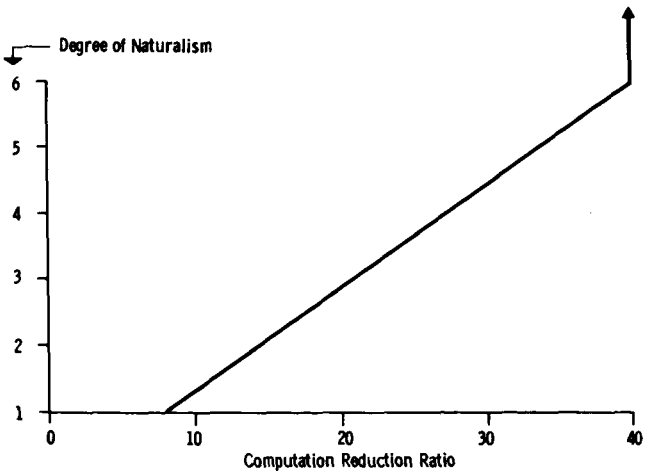


Figure 8. Natural parallelism degree of complexity vs computation speed

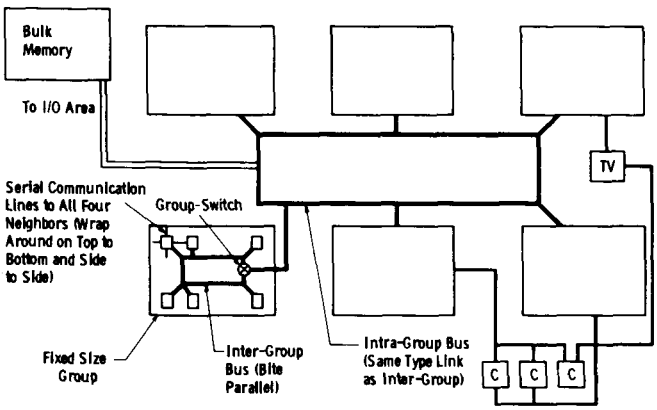


Figure 9. Distributed processor organization

group, shown in Figure 10, can operate either as a controller, an operating cell, or a storage cell. The controller cell provides the global control for a group by placing macro instructions on the inter-cell bus. The operation cells receive these macros from the controller or from their own storage registers, decode them and use them to read out a sequence of operations from the microprogram storage in a cell. The sequence of instructions from the microprogram storage cause storage registers and control registers to be added, exchanged, transferred, etc. Group switches are provided which act as lock-out switches for the particular group during critical phases. During critical phases the switch is set such that any given group will only accept commands and communicate over one of the two inter-group busses. This enables isolation of failures and reconfiguration within the 5-second time constraint.

EVALUATION

The following are some of the considerations in evaluating the three computer mechanization approaches. The Multiple Computer approach provides minimum

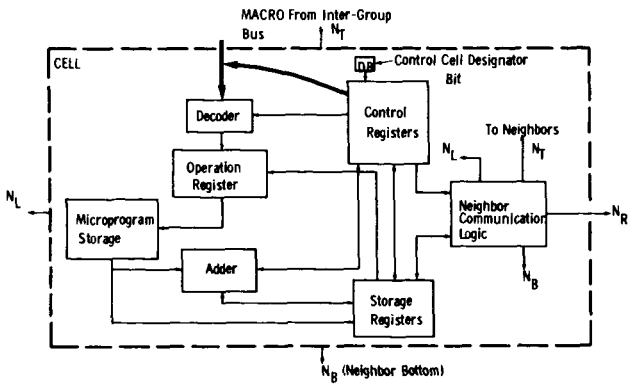


Figure 10. Distributed processor cell

components and communication lines and provides a reasonably good match of hardware to the requirements. The fault detection to a computer is relatively simple; however, fault detection to a level lower than the computer is difficult to meet. Changing computational requirements, it is also necessary to vary computational capability in terms of computer modules. The multiprocessor approach again gives a good match to the requirements and permits localization of failures to modules due to the full intercommunication capability. It also presents the possibility of providing spares at the module level. Down time during reconfiguration after failure is less with this approach and it is possible to withstand certain multiple failures. Expansion of this system in memory and computational speed can be done in smaller hardware increments than with the Multiple Computer approach. Some of the problem areas are that the expansion is limited and must be accounted for in the communication area during the design, and that the number of communication lines is somewhat greater than with the Multiple Computer approach. The Distributed Processor approach provides the possibility of very high reliability and low power consumption due partly to the elimination of the main memory. It is possible to take advantage of the hardware by providing many levels of graceful degradation. The Array Type technology will also be taken advantage of in the Distributed Processor approach. It is possible to expand the computer in small hardware increments if the increments have been anticipated and if the packaging is adjusted. The problems with the distributed approach are that it is relatively complex to define optimum macro and micro instructions and that the programming and executive are relatively more complex.

Figure 11 shows a block diagram of the Monte Carlo approach used in this study for performing a reliability simulation. It should be noticed that probability of fault detection was included in the Monte Carlo approach as well as the probabilities of failures for the various components of the system. Figure 12 shows a typical set of curves that resulted from the simulations for the multiple computer system. The curves show three failure rates or MTBF's of the computers 8,000, 16,000 and 25,000 hours; a probability of detection of faults of 0.99 was assumed in the runs and an on/off failure rate of 10 was used. Curves are also included

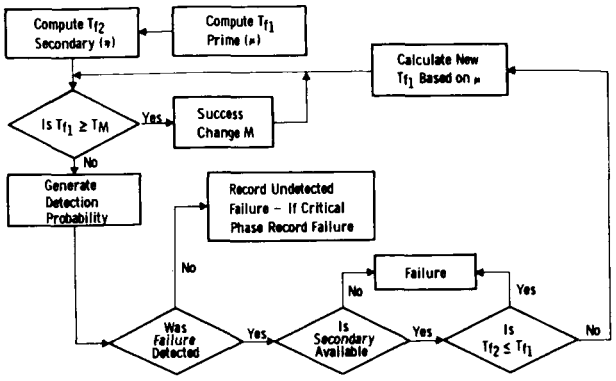


Figure 11. Monte Carlo simulation

which show the effect of having the full memory on at all times during the mission and being able to turn off portions of the memory when not in use. It is seen that a significant increase in probability of success of the computer system is achieved by turning off memory modules when not in use. Other curves have been generated to access the value of failure detection, ratio of on/off failure rates, and computer availability during the mission.

REFERENCES

1. Hokom, R.: "Executive Program Control for Spaceborne Multiprocessors", Spaceborne Multiprocessing Seminar, Boston, Mass., October 1966.

2. Holland, J.: "A Universal Computer Capable of Executing an Arbitrary Number of Sub-Programs Simultaneously", Proc. E.J.C.C., 1959.

3. Slotnick, D., et. al., "The Solomon Computer," Proc. F.J.C.C., 1962.

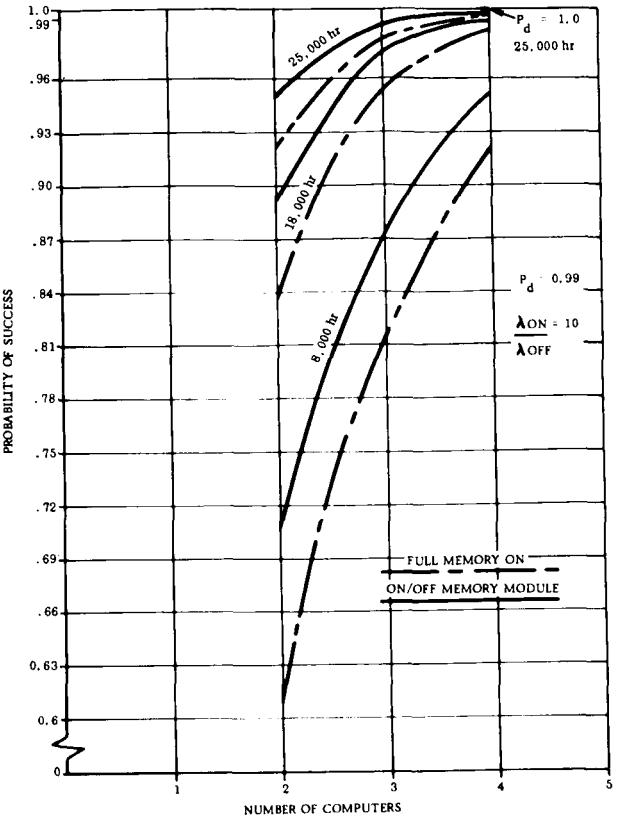


Figure 12. Probability of success vs number of computers

4. "Study of Spaceborne Multiprocessing 1st and 2nd Quarterly Report," Autonetics, Anaheim, California, Sept. 1966.

FUNCTIONAL REQUIREMENTS OF SPACEBORNE COMPUTERS ON ADVANCED MANNED MISSIONS

E. L. GRUMAN

Mr. Gruman received his B. S. E. E. degree in 1960 from the University of Maryland and his M. E. E. degree in 1962 from New York University. Mr. Gruman was with the Bell Telephone Laboratories from 1960-1963 and has been with Bellcomm, Inc., since 1963. With Bellcomm's Computer Technology Department Mr. Gruman has been engaged in on-board checkout and data transmission studies. He is a member of IEEE, Tau Beta Pi, Eta Kappa Nu, and Phi Kappa Phi.

P. S. SCHAENMAN

Mr. Schaenman received his B. S. degree in 1961 from Columbia University; another B. S. degree, also in 1961, from Queens College; an M. S. degree in 1962 from Stanford University; and an E. E. degree in 1963 from Columbia University. Mr. Schaenman has been with Bellcomm, Inc., since 1963 where he has been engaged in the study of overall data flows and spaceborne computers for manned spaceflight. He is presently supervisor of the Computer Systems Studies Group. He is also a member of Phi Beta Kappa, Tau Beta Pi, Eta Kappa Nu, and ACM.

N 67-17103

FUNCTIONAL REQUIREMENTS OF SPACEBORNE COMPUTERS ON ADVANCED MANNED MISSIONS

By E. L. Gruman and P. S. Schaenman

Members of the Technical Staff
Bellcomm, Inc.
Washington, D. C.

SUMMARY

This paper discusses functions which will require support from the on-board computer system during advanced manned missions. In addition to present day functions, such as guidance, navigation, attitude control, etc., the computer system would provide capability for: 1) monitoring, confidence testing, and diagnostic testing for spacecraft subsystems and experiments; 2) inflight crew training with simulations; 3) control and data management for experiments; 4) displays for flight and experiment operations; and 5) G&N of unmanned probes launched from the mother craft.

It is concluded that: 1) The functions (G&N, attitude control) which originally justified using on-board computers are no longer the pacing factors in determining many system characteristics; 2) Mission complexity will force the crew to make extensive use of computer system support; 3) The growth of computer usage in spaceborne scientific experimentation will parallel the historical surge evident in ground-based experimentation; 4) Increased functional requirements will result in a greatly increased number of I/O channels, increased high speed memory, the addition of off-line bulk storage and more powerful processing capability, regardless of the specific system configuration; and 5) The amount of on-board software required for a manned flyby mission will be large relative to manned missions heretofore.

Introduction

In the Apollo program the spacecraft computers are used for the functions of: guidance, navigation, attitude control, operation of simple displays, astronaut-computer communication, and computer-ground communications. They also run tests on themselves and the G&N system. Beyond Apollo, the increasing complexity of missions, and advances in computer technology, will undoubtedly result in a lengthening of the list of functions.

This paper discusses various functional requirements on spacecraft computer systems for advanced manned missions. A planetary flyby mission shall be used for the purpose of this discussion. Nevertheless, the discussion will be applicable, to varying degrees, on long duration earth orbital, planetary landing, double flyby, and other manned missions.

Emphasis is placed on requirements which are new, i.e., not expected to be found on missions through Apollo. It is assumed that the spacecraft must be capable of entirely independent operation, regardless of whether the spacecraft or ground has prime control for the various mission operations. No attempt is made to delineate a specific computer system configuration, although certain gross system characteristics can be inferred.

EXAMPLE MISSION

The planetary flyby mission used as an example here is assumed to begin with assembly and checkout of a spacecraft and injection vehicle in earth orbit. The spacecraft includes a large Manned Module (MM) in which the astronauts normally carry on their activities during the trip, and a small Earth Entry Module (EEM) for the final return to Earth. After injection toward the planet, a few midcourse corrections are made. In transit, experiments in space physics, behavior, and physiology are conducted. Astronomical observations are made using a large telescope. A few days before planetary encounter, several (about six) unmanned probes are ejected from the spacecraft and guided toward the planet. The probes may include orbiters, slow-descent atmospheric probes, and soft landers. The probes communicate at high data rates with the mother spacecraft for a short time before and after periapsis.

On-board the spacecraft, remote measurements of the planet are made using various portions of the electromagnetic spectrum. A large number of high resolution pictures are taken using the large telescope. Data is transmitted to Earth at rates up to one megabit per second from injection until a few weeks after encounter. The maximum rate diminishes to a low of approximately seventy kilobits per second. This low rate lasts for about a month, then returns to one megabit per second. There may be a period on the return leg when the sun lies between the spacecraft and Earth; this would cut off communications with Earth for as long as two months. The return leg of the mission is used to transmit data collected during planetary encounter to Earth and to perform experiments similar to those on the outgoing leg. Earth entry will occur some one and one-half to two years after injection.

REQUIREMENTS FOR FLIGHT OPERATIONS

Monitoring and Testing On-board Systems

Three levels of testing will be required for conducting flight operations: monitoring - an essentially continuous check of certain system parameters; confidence testing - a more detailed check of system parameters before certain crucial events; and diagnostic testing - a still more detailed check of a system whenever it is found to be faulty.

Checkout at these levels is not restricted to inflight needs but is required as well in certain phases prior to launch. Use of an integrated testing concept(1) in which certain prelaunch and inflight tests are carried out in a common manner

by essentially the same automatic equipment appears desirable, both for economic reasons and to maintain continuity of testing. The on-board computer system is the natural candidate for the job.

There are other arguments for using the on-board computer system for checkout. Monitoring system status for long periods of time is a job done poorly by humans and, even worse, is a waste of precious resources.(2) Confidence and diagnostic testing, though higher order tasks than monitoring, are also candidates for automation in order to obtain faster testing with less chance of human error. This is particularly true at hectic times in the mission or if multiple failures occur.

The use of a computer for testing provides the storage media, computational capability, and logical capability for making comparisons and indicating trends with both accuracy and repeatability. Furthermore, the test points and much of the checkout software will already exist from prelaunch requirements. It will also be desirable to have an automated system on board which can assist the ground in determining the status of the spacecraft, especially if part of the crew becomes incapacitated.

Astronauts will control the automated checkout system via a checkout station which will have a keyboard, displays, and communication link with the central computer system. Ordinarily, only summaries of system status will be presented on the displays. Upon request, the astronaut will be presented with more detailed information on any system. He will be able to ask for present, former, or nominal values. He will also be able to initiate confidence and diagnostic tests. His overall capability will be somewhat like that at launch system consoles during an Apollo countdown. The MADAR (Malfunction Detection and Recording) System, an automated inflight checkout and maintenance system being developed for the C-5A transport,(3) is another precursor of the type of system envisioned.

In Apollo, the ACE (Acceptance Checkout Equipment) spacecraft test points are automatically checked on the ground. Only a restricted set of these points is used on board because of the very limited use of inflight maintenance. In contrast, an interplanetary spacecraft will probably have all its "ACE" points available in flight as well as preflight. Some of the points used for diagnostics and all of those used for monitoring and confidence testing will probably be wired into the automated checkout system. The rest--those of improbable use due to limited system usage or lack of criticality--will be accessible by being plugged into a portable interface with the computer.

How many test points will there be? On one hand, the increases in system size and sophistication and the addition of new services will tend to increase the checkout requirements over Apollo. On the other hand, the increasing capability per unit size of electronic devices and other system building blocks will tend to reduce the overall number of points to be tested. The authors' speculate that for a mission such as the 70's flyby example, one can expect a factor of two to five increase over the number of Apollo ACE CSM test points. This implies approximately 2000 - 4000 test points for the MM systems. Diagnostic test points would make up about three fourths of the total number.

Three broad classes of diagnostic approaches are foreseen: 1) those for digital systems (automated); 2) those for analog systems (automated); and those for basic building blocks (semi-automated). Just as they require more test points, digital equipments generally require more complex diagnostic routines than do analog equipments since they are generally capable of many more operational states. Thus, elaborate diagnostic routines are foreseen for systems such as up and down data links, the computer system itself, and the computer interface equipments. Certain digital/analog hybrid subsystems will also require rather elaborate diagnostic routines.

The totality of diagnostic programs will require a significant amount of bulk memory space. For example, the Apollo ACE programs use tens of thousands of words for checkout of a lunar landing mission spacecraft. A planetary flyby spacecraft would require somewhat greater storage for the totality of its checkout programs.

The discussion thus far applies primarily to an MM where the crew normally carries on its activities. It is recognized that an EEM would be checked out prior to interplanetary injection and prior to Earth landing. Its checkout equipment may be partially self-contained, partially contained in the MM. The MM computer system may therefore have to bear part of the load of EEM checkout as well as its own. However, for most of the mission, EEM checkout will not ordinarily be of concern.

G&N, Abort Guidance, and Digital Autopilot

Guidance, navigation, and abort requirements obviously depend heavily on the type of mission involved. For planetary flyby missions, the G&N computation requirements would not be substantially higher than those required for a lunar flyby, except in the vicinity of the planet, where probe guidance (briefly discussed later) is necessary. There would be opportunity for early return by abort only while the spacecraft is significantly influenced by the earth's gravitational field, which is less than one percent of the mission duration. Abort G&N algorithms would be of the same order of complexity as the abort algorithms used for a lunar flyby.

A significant impact upon advanced computer systems could result from the use of strap-down

inertial measurement units (IMU's), which are candidates for planetary mission use because of reliability per unit weight and power considerations. There are indications that strap-down units could increase the on-board G&N computation load by as much as a factor of five to ten with respect to the calculations required using a gimbaled IMU, although use of a digital autopilot tends to lower this factor slightly. Both gimbaled and optical platforms are also possibilities. Either could well be used, if not in a prime role for the entire mission, then as prime for a particular mission phase or as backup.

Displays

Currently, the interior of a spacecraft resembles an airplane cockpit: a profusion of dials, lights, and switches, each with a unique function. For the most part they are connected to sensors with little or no information processing en route. Pilots eventually learn to live with this display jungle, though non-pilots are usually staggered by it. The situation could get worse with the more numerous and complex systems expected on advanced missions.

One source of relief would be to display less subsystem data with the aid of the previously discussed automated checkout system. Another approach would be to combine information from various sensors into integrated situation displays like those recently developed by Army-Navy research for aircraft use.[4] In one such display system, data is collected from the gyros, radars, air data computer, compass, instrumentation landing system, and fuel flowmeters. The central digital computer system processes the information and provides the outputs to run a vertical situation display and a horizontal situation display.

A version of the vertical situation display, made by Kaiser Aerospace and Electronics Corporation, is currently operational in the Grumman A-6A Intruder. This "contact analog" display shows the command flight path as a highway in the sky. The highway is in proper perspective as viewed from the current position of the aircraft. The pilot flies his command course simply by trying to stay on the highway. Other features include a distance scale, the aircraft attitude in three dimensions, and symbols for a target and a weapon release point.

In a second mode, the display projects a synthetic 3-D view of the terrain ahead of the plane, using range, altitude and azimuth information from the radars. The terrain is shown as ten vertical slices at various ranges (1/4 mile ahead, 1/2 mile ahead, etc.). Each slice shows terrain height vs. azimuth at that range, so that the overall effect is one of looking at a three dimensional contour map. Tests show that pilots can follow terrain contours of the radar display more accurately and with more confidence than with visual references in clear weather. The radar sharpens terrain features, and accurately measures range which the eye only estimates. Pilots like these displays.

The flight display puts a light incremental computation load on the central computer, since most of the displayed data must be calculated regardless of the type of display used. The amount of additional computer memory space which may be charged to this particular display is estimated at less than 1000 words.

It seems reasonable to anticipate variations of the above integrated situation displays which would assist rendezvous, earth entry, attitude control, and virtually all other piloting function aboard a manned spacecraft. For example, consider a manually controlled rendezvous with another vehicle which has an extremely unfavorable lighting background. A display showing the vehicle in perspective, some range markers, the desired rendezvous trajectory, and appropriate command information would be of considerable aid to the pilot and thereby increase reliability in a critical situation. Other integrated displays might be used for projecting entry corridors as three dimensional paths, or in simulations used for on-board crew training.

In addition to these somewhat exotic displays, there will be more mundane CRT or electroluminescent (EL) displays for showing such things as X-Y plots and alphanumerics. For example, the automated checkout system will use these displays. Others will be associated with experiment control and data management. The Apollo spacecraft uses numeric EL displays for showing selected outputs from the AGC.

The role of the computer in all of these various on-board display systems is obvious: it collects and formats information from various sensors; stores and fetches data; performs necessary computations; and composes appropriate data into the various presentations by commanding the appearance and positioning of symbols, waveforms, and other types of graphics. In spite of the inference one might draw from the vertical situation display example presented here, the load on the computer for driving displays may vary over a very wide range.

The MTBF's of present integrated situation displays are estimated at several days to several weeks--too low for planetary missions. Anticipated improvement in CRT and/or EL technology will significantly increase these MTBF's. The fact remains that most of the elements of the present display jungle have the inherent reliability advantage of the simple over the complex and of not putting all the eggs in one basket. Until the more sophisticated displays are proven reliable, they will probably be backed up by a reduced set of "simple" displays.

Astronaut-Computer Communications

The Apollo astronauts communicate with the Apollo Guidance Computer (AGC) using a keyboard

with buttons for: numbers 0 - 9, +, -, VERB, NOUN, CLEAR, STANDBY, KEY RELEASE, ENTER, and RESET. The astronauts consult a code book and punch in digits representing verbs, nouns, or data. The verbs are simple commands to the computer, such as "display (noun)", "monitor (noun)", "load (noun)". The nouns are various parameters such as velocities, angles, rates, positions, time, etc. The computer communicates with the astronauts via a simple numeric EL display and a set of status lights. Almost the entire computer-astronaut dialogue centers around guidance and navigation.

The expanded set of computer functions on advanced missions will require more frequent, more inclusive, and more sophisticated man-machine communications than in Apollo. A premium will be placed on speed and accuracy of communications and on minimizing the tremendous learning burden of the astronauts. The improved displays suggested in the preceding section will be one source of aid. Another will be higher level input languages. Research is needed to define either a general spacecraft command and control language or a group of function-oriented languages for the astronaut-computer dialogue. Reasonably detailed diagnostics of input programs should also be provided. A high level language package plus increased diagnostic capability imply the availability of additional high speed memory space.

Another improvement will be to enlarge the keyboard and have individual keys for frequently used words, with less frequently used words inserted via a general set of alphanumeric keys. The choice of words to be considered "most frequent" might be left to the user and allowed to vary from person to person and station to station. All commands could still be initiated from any keyboard.

Optional hard copy of inputs or outputs would be another desirable feature. Also, punched or magnetic cards might be used for storing frequently used programs. The card would be inserted into a computer input device, similar to card dialers used with telephones, in lieu of punching buttons whenever the program is desired. Voice input devices might also become feasible for some portion of the input repertoire.

Inflight Crew Training

"During a several month interplanetary voyage, crew members could lose some of the skill they have developed in such maneuvers as earth atmosphere reentry."(5)

Alan B. Shepard
February 19, 1964

Planetary flybys will put at least one to two years between the time an astronaut last flew (or practiced in a full scale simulator) an earth entry and the time he must do so again. The intervening time plus the physiological and psychological demands of the mission will tend to degrade his ability to perform the task. Various other on-board tasks will have smaller but still significant intervals between practices. Examples are planetary photography at close range, targeting and guidance of probes, and on-board planetary encounter experiments.

The astronauts must somehow maintain their skills in these tasks, either by live practice runs or simulations. Ideally, they should use the same controls, displays and systems for practicing as in actual operational usage. However, there is a school of thought that one should not take flight-critical controls and displays off line for simulations during a mission. Moreover, introducing simulation modes (with switches and additional input and output paths) may lower system reliability.

We therefore anticipate the existence of an on-board training, simulation and behavioral research station. It would have displays and controls which can assume different configurations to simulate different crew stations. It would also use the computer system for controlling real time simulations, storing norms, simulating certain systems, evaluating results, and compiling subject profiles. The facility would have the following uses:

1. Training. This is required to maintain crew skills which are not frequently used.

2. Crew reassignments. Each astronaut will be a specialist in some areas and cross-trained in others. At some point in the mission, perhaps due to the incapacitation of some other crew member, it may be desirable or necessary to reassign an astronaut from his original specialty to another. The equipment and software used for "routine" training might suffice, though some additional "teaching machine" capability may also prove to be desirable.

3. Checkout of new procedures. These may be established by the flight crew or ground during the course of the mission [The mission will be of such duration that even state-of-the-art advances are possible.] This function requires the ability to insert large new programs into the computer from ground or smaller ones from on-board.

4. Behavioral research. In addition to the biomedical monitoring of the astronauts, certain behavioral studies will take place. These will consist of tests of reaction-time, decision-making and problem solving. The results of these tests will be correlated with biomedical data to indicate the "condition" of the astronauts at various points in the mission. Since these behavioral studies will require the use of displays and

controls, it should be possible to use the facility for behavorial experiments as well as for simulation and training.

The simulations used in conjunction with this facility would be major users of computational time when running. They could be among the largest programs on board. The Apollo Mission Simulator and LM Mission Simulator programs each run greater than 100K words. Though not necessarily representative, they indicate how large these simulations can become.

EXPERIMENT REQUIREMENTS

On-board Experiments

"Nuclear instrumentation is undergoing revolutionary changes because of [the] rapidly increasing use of stored-program computers by experimentalists in nuclear-structure laboratories."

John V. Kane

"In the high energy physics laboratory the most remarkable development that has occurred in the last five years has been the introduction of the digital computer as an active part of the experimental apparatus."

George W. Tautfest

Both of these quotes were taken from the July, 1966 issue of "Physics Today" and indicate the effect computers have had on ground-based experimentation. It is likely that spaceborne computers will have a similar effect on space experimentation within the next decade when one considers that they have been virtually unused thus far and that the number and complexity of experiments are increasing. For example, the particles experiment on Explorer I measured omnidirectional intensity of particles of any type. On OGO-E, the particles experiment will measure directional characteristics and intensity as a function of particle energy and type.(6) Several uses of computers in on-board experimentation are suggested in the following sections.

Experiment Checkout and Calibration -- It is estimated that there will be about 30-40 major pieces of experimental equipment on-board a flyby spacecraft in addition to about forty carried in the unmanned scientific probes. There would also be a large (40") telescope with its own attitude control, photographic and TV systems.

About one third of the on-board experiments and the telescope system must be monitored and occasionally tested or calibrated throughout the mission, much in the manner of spacecraft systems. Another third of the on-board experiments, the forty experiments carried in the probes, the various subsystems of the probes, and the flyby photography and TV systems must all be tested, and calibrated if necessary, shortly before planetary encounter. This may involve on the order of one to two thousand test points.

The checkout and calibration tasks should be automated to the fullest extent possible for reasons similar to those given for automating spacecraft systems checkout (obtain speed and accuracy, reduce crew workload, avoid human error, etc.). The precision possible with a computer system will be of even more importance for these tasks than for testing of spacecraft systems because of the generally greater precision required by scientific measurements compared to operational engineering measurements (G&N systems excepted).

As in testing, the use of the computer system for calibration will permit the use of complicated or exhaustive schemes which might not otherwise be used. This, plus the accuracy and repeatability of the computer, the ability to record steps in the calibration process, and the presence of man, will result in greater confidence in the calibration of the experiments--an important advantage over present experimentation.

Experiment Control -- At various times in the mission experiments must be turned on, have their sensors exposed, be run through a warm-up sequence, be coordinated with other experiments, undergo cyclic changing of operational modes, etc. Although most of these functions could be implemented with simple programmers, they are candidates for computer control in order to allow flexibility in flight. Building a programmer with enough flexibility to arbitrarily change the timing and sequencing of experiments may be less desirable than building a wired interface with the computer system, using a modest amount of computer time and memory space, and keeping the flexibility in the software.

A more complex type of control than sequencing may be needed for experiments such as patrolling for solar flares. In this experiment, the sun will be monitored for about half of the planetary mission using telescopes, X-ray, UV, visible and other electromagnetic sensors, cosmic ray and solar proton sensors, etc. While the crew may occasionally or even regularly monitor the sensors, it does not seem reasonable to spend a man-year for this purpose, since the flares occur only about 0 - 20 times per year. Nor can one rely on earth to warn that a flare is occurring. The visible portion of a flare lasts from several minutes to one hour, whereas the communication time for transmitting the warning from earth to spacecraft will be in the order of 0 - 30 minutes; the flare or its beginning might be missed entirely.

It would be economical in film, bit storage, and man-hours to have a system which samples the sensors at a low rate until something unusual occurs, then alerts the astronauts, increases the sampling rates and photographic repetition rates, and brings on-line any sensors which may have been

inactive. The computer system would be used to discover the "something unusual," perhaps using pattern recognition techniques to determine an unusually bright area on a TV picture of the sun. The computer system would command the initial response. This overall scheme represents one form of data compaction by computers.

Several experiments, including the above, will require accurate pointing and holding to targets on various heavenly bodies. Calculations for these functions may require data from the onboard autopilot and G&N system, the telescope attitude control system and ephemeris tables.

Experiment Data Management and Displays -- The astronauts themselves will have an important role in the checkout, calibration and routine control of the experiments. In addition, they should have some capability to take advantage of discoveries, explore anomalies, redirect experiments in case of failures or mistakes, and select data for transmission to the ground.

To perform these tasks the astronauts must be able to sample, in real or near-real time, data from any of the experiments being conducted. Though this may not always be possible, it should serve as a goal. They must then be able to process the data with the aid of the computer system if necessary. Processing may involve curve-fitting, computation of statistics, statistical filtering, correlating data from several experiments, solving systems of equations, and an extremely wide range of other possibilities.

Both raw and processed data should be capable of being displayed in a variety of alphanumeric and graphic formats. Display options should include symbols, histograms, X-Y plots, waveforms, and scatter diagrams. The astronauts should also be able to display data from several related experiments simultaneously and to request a priori expected results to be displayed alongside actual results.

Probes

In our example mission, the crew of the flyby spacecraft must check out and count down the approximately six scientific probes about one week prior to encounter, using the computer system for automated testing and sequencing. About three of the probes will be of the complexity of Lunar Orbiter or Surveyor; the other three will be relatively simple atmospheric probes. All would be launched within a period of a few days.

After injection toward the planet, the probes will be tracked from the spacecraft by radar and optical techniques. Before landing (or going into orbit), the probes receive one or two midcourse corrections from the spacecraft. The corrections are based on the continually improving knowledge of the spacecraft's trajectory relative to the planet. The trajectory improvement is based on inputs from the on-board sextant and telescopes and from earth-based continuous tracking.

Thus, the spacecraft must act as a spaceborne tracking and flight control facility while at the same time navigating for itself. It has been estimated that the spaceborne computation load for probe guidance will be several times that of the Apollo LM descent guidance, the most demanding of the Apollo guidance programs.

It is not clear to what extent guidance computations for the various probes would overlap in time with one another and with other tasks. It is clear, however, that the probe guidance tasks will be of extreme importance and will be demanding attention at one of the busiest, most critical times in the mission.

RELIABILITY REQUIREMENTS

The heavy dependence upon the computer system suggested in this paper presumes extremely reliable hardware and software. In Apollo, the guidance computer is required to have a certain MTBF. A computer system for interplanetary missions will also have MTBF requirements, but in addition will be required to absorb (at various levels) certain malfunctions without hindering performance, and certain other malfunctions without complete loss of performance. Considerable study is being given to hardware reliability, and techniques are rapidly becoming available to meet these enhanced requirements on hardware.

The AGC software is required to produce appropriate outputs under allowable input conditions. This general software "quality" requirement, already difficult to achieve, will tend to become even more elusive in future missions. The housekeeping functions introduced by redundancy and switchable configurations as well as increased I/O will significantly complicate the package of programs relative to the AGC software.

In studying the software quality control problem of Project Apollo, it was concluded that the use of strong management procedures including tightly controlled documentation was the most valuable approach available.[7] Producing detailed software documentation has a most desirable by-product, that of forcing the thinking out of program possibilities. This tends to eliminate problems due to logical inconsistencies. It appears that strong management control will remain a valuable approach for interplanetary missions. The general problem of software reliability is ripe for new approaches.

CONCLUSION

The following inferences can be drawn from the above discussion:

1. The functions which originally justified bringing a computer on board a

space vehicle (G&N, attitude control) will no longer be the prime factors in determining characteristics of the computer system. The new functions, automated checkout, flight crew training, expanded displays, and on-board experiment control and data processing, will force the future on-board computer system to have the capability to efficiently perform character manipulations as well as mathematical calculations.

2. The complexity of spacecraft flight operations and scientific activities will alter previous mission management concepts. The crew will make extensive use of the automated system centered around the computers. This will require highly efficient astronaut/computer communications. More sophisticated displays and input languages than those being used in Apollo will be required to accomplish this.

3. The growth of computer usage in on-board experimentation will parallel the rapidly expanding usage in ground-based experimentation. This trend will impose significant requirements on both the memory and processing capability of future on-board computer systems.

4. It is estimated that the increased functional requirements will result in a greatly increased number of I/O channels, an increase in high speed memory of an order of magnitude, the addition of off-line bulk storage memory capability, and a more powerful processing capability than is presently available in Apollo. These characteristics will require either a very fast central processor or a multiprocessor system.

There are various functions in addition to those discussed in this paper which may be candidates for use of the on-board computer system. For example:

. Housekeeping, such as automatic balancing of solar heat loads by attitude control.

. Communications management, such as routing messages between spacecraft systems, experiments, probes and ground, and pointing spacecraft antennas.

. Medical diagnosis

. Data reduction and compression, in addition to controlling experiment sampling rates as in solar patrols. (This is not foreseen as a vital function because of the expected 10^6 bits/sec. transmission capability on board.)

As the old saying goes, further study is needed.

ACKNOWLEDGMENT

The authors wish to thank the many members of the technical staff of Bellcomm who participated in discussions on this paper.

REFERENCES

(1) Greene, D. W. and Wood, E. C.: On-board Checkout System Concept. Stepping Stones to Mars, AIAA/AAS Volume of Technical Papers, March, 1966, pp. 263-268.

(2) Chase, W. P.: Integrating Crew Performance into Space Vehicle System Design for Optimum Reliability. Manned Space Reliability Symposium, AAS, June 9, 1964, p. 46.

(3) Stambler, I.: The Big New Transports. Space and Aeronautics, Vol. 46, No. 3, 1966, pp. 54-62.

(4) Evanzia, W. J.: "A View from the Cockpit". Electronics, August 22, 1966, pp. 145-148.

(5) Shepard, Alan B.: "Training by Simulation". Smithsonian Institution, Edwin A. Link Lecture, First, Washington, D. C., February 19, 1964. Published in International Aerospace Abstracts.

(6) Bostrom, C. O., and Ludwig, G. H.: "Instrumentation for Space Physics". Physics Today, Vol. 19, No. 7, July, 1966, pp. 43-56.

(7) Liebowitz, B. H., Parker, E. B. III, and Sherrerd, C. S.: Procedures for Management Control of Computer Programming in Apollo. TR 66-320-2, Bellcomm, Inc., September 28, 1966.

DESIGN CRITERIA FOR A SPACECRAFT COMPUTER

DR. RAMON L. ALONSO

Dr. Alonso is Assistant Director of the Instrumentation Laboratory and Lecturer in the Aeronautics and Astronautics Department at Massachusetts Institute of Technology. He is a designer of digital systems used in guidance and navigation equipment; his present concern is the design of compact memories for airborne computers, and mechanization of logical and electrical design procedures.

Dr. Alonso was born in Buenos Aires, Argentina. He came to the United States in 1947 and attended Harvard University where he received his A. B. degree in Physics in 1951, his M. S. degree in Mechanical Engineering in 1952, and his Ph. D degree in Applied Mathematics in 1957. He was at Bell Telephone Laboratories during 1952 and 1953.

Dr. Alonso joined the Digital Development Group of M. I. T. 's Instrumentation Laboratory in 1957 and was appointed Lecturer in 1964.

DESIGN CRITERIA FOR A SPACECRAFT COMPUTER

By Ramon L. Alonso
Albert L. Hopkins, Jr.
Herbert A. Thaler

M.I.T. Instrumentation Laboratory
Cambridge, Massachusetts

SUMMARY

Extrapolation of Apollo experience to spacecraft computers of the next generation indicates a need for digital systems of greater computing and interface activity, and of greater reliability, than has been realized to date.

An idealized collaborative multiprocessor structure in which a number of processing elements are tied together by means of a single multiplexed data bus is explored. At least one job assignment procedure is possible for which no one processor has to act as 'master', and which can survive processor malfunctions or the deletion or addition of processors to the bus, thus accomplishing 'graceful degradation' and 'reconfiguration' of sorts. The single bus structure as used here implies things about compilers for it, and also certain bandwidth relationships between processors, bus and common memory. Rough estimates based on short extrapolations of circuit technology show that the structure is probably realistic.

DESIGN TRENDS

Introduction

In manned spacecraft to date, more uses have been identified for on-board data processing than could be provided by the computers therein. Computer designers are inclined to anticipate this sort of problem by their natural tendency to supply greater performance than the application seems to require, but have been inhibited in the spacecraft area by apparently inelastic size, power and reliability constraints. These constraints are relaxed when it is discovered that mission success is imperiled by lack of adequate computer performance. This very likely arises at a time which is too late to reconfigure the computer within the mission schedule. Instead, mission objectives are apt to be restricted and a large software effort is mounted to prepare and verify programs which squeeze out maximum performance. A lesson for the next spacecraft generation is that graceful expandability should be a fundamental requirement for the data processor and other systems. This can result in the ability to profit from lessons learned in the development phases of a mission by reconfiguring the on-board systems with a minimum of impact upon the spacecraft.

In this paper, we review some general requirements for the next spacecraft computer generation and the forecast for hardware available in the coming years. In the absence of the development of a suitable self-organizing automaton, the multiprocessor structure appears to be best suited to both the requirements and the hardware available. We describe an idealized multiprocessor organization and examine its performance in terms of the performance of its components.

Multiprocessors

Extrapolating the Apollo mission to a planetary mission has many pitfalls, as entirely new problems and solutions are involved. From the computer's point of view however, the requirements can be expressed independent of many of the attributes of the total spacecraft. Size and power constraints should not be expected to be much different from what they are today. However, reliability over a period of several years adds a new dimension to the problem, for in a system of perhaps millions of solid-state electronic elements, it must be assumed that several, perhaps many, will become inoperative either due to poor quality or to severity of environment. What is needed is a system whose performance will not be reduced below the minimum required for survival of the spacecraft, unless failures of calamitous proportions occur. A new concept of graceful degradation has arisen to supplement the old notion of redundancy in which elements may fail, but the circuits which contain them continue to function with no degradation. If more elements fail than the redundancy can cope with, the circuit will fail, and with it, the system. Graceful degradation implies an organization in which circuit failure reduces, but does not suppress, the machine's throughput. The brain has this characteristic, but neuron-based automata have not yet exhibited promise for miniature control computer applications.

In a multiprocessor organization, graceful degradation and graceful expansion are related properties, both made possible by the independence of the constituent functional units: processors and memories. A multiprocessor is more complex and expensive than a like sized array of independent computers. Its value is greater, for its performance depends on the number of units functioning at any time. To increase the power of the machine, processors and memories can be added without affecting parts previously present and, at least equally important, without affecting existing programs. Each processor may be made as powerful as the technology allows, but in the face of the reliability problem, it appears more desirable to build simple, reliable processors in greater quantity so as to minimize the impact of a single processor's loss.

The multiprocessor structure is compatible with several of the requirements of the spacecraft application beside that of reliability. For one thing, communication between the multiprocessor and all other spacecraft systems can be handled in the same fashion as communication among the processors, thus affording a unified treatment of the problem of input-output involving perhaps hundreds of external functions. In a time-multiplexed serial

transmission structure, for example, a new system can be added to the multiprocessor's interface with virtually no more changes beside the addition of access lines for the new system to the coaxial cable (or waveguide) run. Today, multiwire cable and connector problems probably constitute half the battle in making spacecraft systems work.

Another example of the multiprocessor's well-suitedness is the natural division of many spacecraft data processing tasks into short jobs of fractional second duration. This is a result of the multiplicity of independent programs serving the many systems involved, and also of the sampled nature of control computations. Each program typically has a low duty cycle, requiring brief service several times per second. Each instance of service can be treated as a separate job to be handled by any available and competent processor. In the Apollo spacecraft, repetition rates for jobs vary from a few tens per second down, with no more than eight jobs running at a time. In future we can expect on the order of a hundred programs running at once and tens or hundreds of samples per second per program.

Hardware

Regardless of what organization may be used, increased performance without increased size can be obtained only with smaller and/or faster components. Size is the key to speed by virtue of the finite velocity of information transmission and of the power (hence size) of an element which drives a long (hence reactive) line. The first effect moreover requires characteristic impedance termination to avoid reflections, which further aggravates the power problem. Efforts to shrink components are hampered by the difficulty of interconnecting components reliably in a small volume with adequate yield.

An area in which great progress is being made, with promise of improvement, is in the creation and interconnection of large numbers of semiconductor elements on a single wafer. Within a wafer, signals can be transmitted at a higher rate than from wafer to wafer. Likewise, the propagation delay of an element no larger than required to drive an internal interconnection will be less than that of an element large enough to drive an external line. The designer is challenged by this technology to organize his equipment into local high speed areas, interconnected by as few lines as possible. How to do it depends on the number of elements per wafer that can be realized. If it is hundreds, then we think in terms of arithmetic and error detection circuits, multiplexers, digital-analog converters, sequence generators, scalers, and small scratch pad memories. If it is thousands, then small processors, medium sized scratch pad memories and small associative memories could be made. If tens of thousands or more, possibilities of rather elegant processors come to mind.

In any event, logic is becoming inexpensive, indeed virtually expendable, to a point where using wire, cable and connections to save it is uneconomical. Thus it is anticipated that all spacecraft systems will have local digital circuitry for encoding, decoding, and multiplexing information for transmission in a common language to the computer and elsewhere. One of the outstanding jobs of the computer designer is to coordinate with the manufacturers of large integrated semiconductor circuits to best exploit this new technology.

Memory will be of several types to serve the various functions of scratch pad, data storage, and program storage, either in a common area or associated with a given processor, or both. Enough separate memories with separate driving circuits must be supplied to meet the graceful degradation criterion, and enough words must be supplied in each memory to do the job. Scratch pad memories might be from 2^7 to 2^9 words; common erasable storage will perhaps require 50 words per program, or more than ten thousand words in all. Program memory would have on the order of a thousand words per program, hence hundreds of thousands of words in all. All three sizes are an order of magnitude beyond Apollo without even considering the additional cost of redundancy. In the light of the growth of computer sizes and requirements in the last ten years these estimates may be somewhat conservative.

IDEALIZED MULTIPROCESSOR STRUCTURE

System Structure

As a model upon which to base our size and performance estimates we use an organization which is simple, yet which contains the elements of a general class of multiprocessors. Starting with a group of processing elements (roughly computers) each of which has its own program and scratch pad data memories, we create a combination in which there is no one supervisory element or processor, but which is truly collaborative.

The first item needed in addition to the processors is an infallible data distributor by which information is transferred among units. A simple form of distributor is a time-multiplexed bus. Every unit having access to this bus can receive all data which appears thereon. Every such unit can also transmit data upon the bus by means of a multiplexer circuit, associated with the unit, which emits the data at an appropriate time. The problem of scheduling time is handled by making each multiplexer enable the next in line as soon as it is through sending data. The next multiplexer will then send its data unless it has nothing to send, in which case it will skip the enable on to the following multiplexer (see Figure 1).

The next item needed is a common erasable memory in which to store data needed to start jobs. This memory must either be infallible, or else have graceful degradation properties of a sort which will be left unexplored in this paper. The memory has access to the bus as do the other units of the multiprocessor. It is interrogated by means of a message sent from a processor specifying its own identity

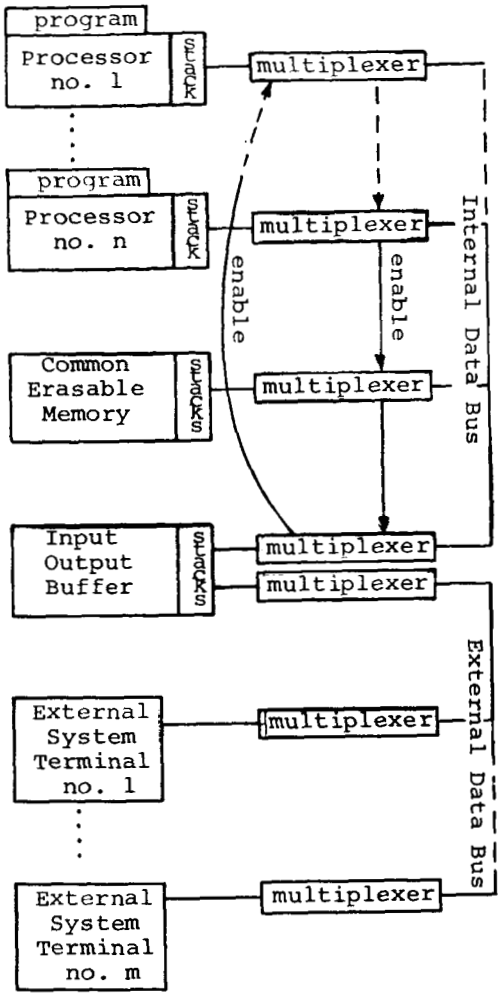


Figure 1. Collaborative multiprocessor model.

and that the contents of memory address k is desired. Upon receipt of this messsage, the memory places it in a waiting stack. When its turn comes, the message causes a memory cycle to be executed, and both address and content to be delivered to another waiting stack for transmission on the bus. The requesting processor will recognize its answer as it appears on the bus.

The last item in the multiprocessor is an input-output buffer unit, capable of relaying messages between multiprocessor units and external system data terminals. Although it is possible in principle simply to extend the multiprocessor bus out to the external units, it is probably preferable to accommodate the external data transfers on a separate bus system. This not only isolates the multiprocessor from its environment for conceptual analysis, but as a practical matter permits the use of different sequencing techniques for the mutually distant remote multiplexers than for the internal, closely packaged ones. Except for this, the remote systems may be considered to be specialized processors, and treated accordingly in the analysis.

Processing Element Properties

The processing elements P are thought of as small general purpose computers with a number of features not normally presumed in connection with processing elements. These are:

Program Storage -- Each processor has its own copy of all programs. The programs are written as pure procedure. This redundant program storage can be dispensed with by having one or several memories which the various processors can interrogate, but it simplifies discussion to have it. In particular, each processor has a list of jobs it can undertake, plus any additional information required by each job, such as starting address, data locations, etc.

Message Transmitter and Receiver -- A processor is connected to the data bus multiplexer by way of a transmitter and receiver section. This section may have a job request stack, as discussed below, and does have means for discriminating among or originating various messages, such as common memory transfers, job requests, job acceptances (see below). An important property is that this section be "infallible", meaning as reliable as we can make it; more to the point, it cannot fail in such a way as to disable the data bus.

Self Error Detection -- Each processor must be capable of diagnosis at least to the extent of detecting any errors within itself. The result of an error in a processor must be a special job request message put on the data bus so as to have each processor inform all others when it malfunctions or when it becomes inactive (e.g., power failure); this is the reason for requiring an "infallible" message transmitter and receiver. Error detection need not be instantaneous; it is probably sufficient to detect errors within a job execution interval and not issue false job results. The detection of certain kinds of errors such as inactivity, or programs becoming "lost", requires either a certain minimum time or else an uneconomical amount of equipment. The area of error detection and/or correction may be one of the more difficult ones in multiprocessor element design.

In addition each processing element has a scratch pad storage, an arithmetic unit and rudimentary interrupt system which will enable single memory cycles out of sequence. The latter should permit a check within several memory cycles to see if a job requested is available in this processor's repertoire of procedures.

OPERATION

Job Assignment

A view of the detailed process of job assignment

is important in ascertaining if the single data bus structure is either possible or desirable, and if graceful degradation will occur.

Definitions

P	Processor
P_1, or P_i	A specific processor
J	Job
J_1, or J_i	A specific job
Y	Priority
Y_1, Y_i	A specific priority $Y_{i+1} > Y_i$
R(J,Y,T)	Job request message
A(J,P)	Job acceptance message
E(J,P)	End of job message
T	Time

The general job assignment can be as follows:

1. R(J,Y,T) appears on the data bus, issued by either a processor or an input-output unit. This is a request to do job J, which has priority Y, and to do it at time T. The time at which the job is to be done can be 'now', or 'as soon as possible', or some specified time in the future.

2. Each P capable of doing J records R, whether busy or not, in a stack with certain associative properties. The messages may be retrieved by keying on J, on T, or on the maximum value of Y. Processors are either free or not. If not, they are doing a job J of a certain priority Y.

3a. Suppose $J=J_i$; when $R(J_i,Y,T)$ appears on the bus the free processors $P_1, P_2 \ldots P_j$ each compose a response message $A(J_i,P_1)$, $A(J_i,P_2)$, ... $A(J_i,P_j)$. Some one of the free processors will have first turn at the data bus (because the bus is time multiplexed) and will issue an A-message. All P, free or not, then elide $R(J_i,P,T)$ and also any redundant $A(J_i,P)$ they may have prepared, and which is waiting its P's turn on the bus. After A is issued by P_j, P_j must bring all pertinent information about J_1 from the common memory into itself.

3b. If there had been no free P, then $R(J_1,Y,T)$ would remain outstanding in all P. All those P doing jobs with lower priority than that of the job requested also prepare response messages $A(J_1,P)$. Again, some processor will be first to issue $A(J_1,P)$ because of the bus multiplexing, and all other $R(J_1,Y,T)$ and $A(J_1,P)$ are annihilated.

The P that undertakes a new J_2 of priority Y_2 higher than the priority Y_1 of J_1 has a choice: it may take on the new job J_2 while keeping all the information about J_1 within itself, if it knows J_2 to be short (such information can be part of the job name itself, or of its priority measure). Or, if J_2 is not short, P must, after issuing $A(J_2,Y_2)$ but before actually doing any work, transfer all pertinent information used by and about J_1 to the common memory and issue $R(J_1,Y_1,T)$. In this way another P can undertake J_1. Common practice is to program jobs with "bump points", which minimize the information that must be sent to or brought from common memory in the event of interruption. The value of knowing when J_2 is short enough to allow the same P that was doing J_1 to resume J_1 after doing J_2 is in the saving of common memory transfers.

4. The end of a job, or the interruption of a job, also requires a message E(J,P).

5. Each A(J,P) issued is recorded in common memory, and annihiliated by the subsequent E(J,P) with matching J. In this way there is at all times a record of which J are being executed and by which P. This information permits restarts in the event of a P failure, as will be discussed below.

6. Jobs to be executed at appointed times are of importance in sampled data systems such as spacecraft. The same stack used for storing unsatisfied job requests can be used to solve the problem. The outstanding job requests R(J,Y,T) may be sorted (or retrieved associatively) by $T_o \geq T$, where T_o is the present time, and further sorted by priority. For each new T_o the stack is interrogated to see if one or more jobs are outstanding. If so, an A-message is prepared, as in 3.

Job Stack

The stack which contains the job request is a potential problem area. On the basis of estimations of system size and speed, and of future integrated circuit sizes, we have guessed the stack size to be 100 words of 50 bits each. The required associative properties might be simulated by circulating the contents of the entire memory in between job requests, and for each increment of time. A recirculation time of the order of a few microseconds looks reasonable from the point of view of circuit technology (10 nsec per bit, for word-parallel shifting). This access time is consistant with a time granularity and a job request interval of the order of ten microseconds, which appear adequate. It is not yet clear, however, whether room for 100 outstanding job requests is enough.

The job assignment and interrupt structure which has been defined previously assumes that

every processor contains a job request stack with associative and comparative properties. In order to avoid the N-tuplication of this potentially expensive stack, the structure can be modified slightly. One "infallible" copy of the stack is maintained in common memory, and is capable of initiating jobs in any processor. The primary difference in the message traffic flow is that a Bump Message [$B(P_i)$] must be defined and transmitted at bump points. Additionally, the bumping option available to the distributed table system which eliminates unnecessary common memory transfers is unavailable to the single table system.

Degradation

The multiprocessor can degrade gracefully if, together with the postulated infallible common memory, the message bus and the part of each processor concerned with message handling are also infallible. It is necessary that a processor failure generate a message, i.e., a job request. The job undertaken by some other processor is to reissue all job requests shown outstanding for the failed processor. Since the input information (the list of outstanding A-messages) is still available in common memory, recovery can be effected by having other P's do the jobs over again.

There are other interesting degraded conditions. One of these is when there is one processor. The message bus then has only one occupant, P_1. When P_1 issues R, P_1 receives it, stores it, computes $A(J,P_1)$ issues it, annihilates R, and gets on with the job. Hence the bus structure must be such as to allow message sending processors to receive their own messages. The single processor will also behave appropriately in the event of a higher priority J_2 appearing while it is doing a job J_1.

General system overload is another case of interest. Suppose the number of job requests becomes large for the system, and the list of R messages stored in each P increases to the point of taxing that stack. If by "graceful degradation" we mean that jobs of higher Y get done first, and that jobs of lower Y get postponed, but done eventually, then we must provide means for making room in the "pending R" stacks. Other strategies are possible, such as proportioned processor occupancy. One way to do this is to have each processor store in common memory (or in its own scratch pad, if it has one big enough) the job requests of lower priority and later time of execution. One interesting point is that, if a processor has many unserviced R's in its stack, other processors are apt to have the same messages in their stacks. Hence, as the lower priority job requests are stored in common memory, a message must be issued for annihilating the same requests stacked in other processors. After making room in the stack the original processor must issue a job request that the demoted job requests now in common memory be reissued.

IMPLICATIONS

Software Considerations

Despite the fact that most of the calculations for a spacecraft are sampled by nature, there exists a substantial programming burden in sectioning programs into jobs of proper length and establishing the packages of data required to shelve and resume the program for interruption and restart. This burden cannot be placed on the programmer because, as a practical matter, computer users do not (and should not have to) know very much about the computer they use. The onus clearly falls upon a compiler. Programs written as a single job must be segmented automatically so as to be able to restart and permit efficient interruption. Writing such a compiler probably represents a task of the same order of magnitude as the design of the multiprocessor itself, and also represents an advance over present compilers. The above multiprocessor design (and very likely, any other) would not be attractive without either the prior existence of a suitable compiler, or knowledge that one can be written.

An interesting extreme form of program segmentation into jobs consists of letting each job be an instruction of an elementary type such as multiply (or perhaps as complicated as a floating point vector operation). The job name must in this case contain data addresses and a next instruction address; or else the job name can be simply the address of an instruction. This would undoubtedly result in inefficient processor usage, but it might lead to useful segmentation techniques.

Estimates of Performance

An order of magnitude estimate of performance requirements for this ideal multiprocessor can be derived from an extrapolation of Apollo experience. Within a few years time we shall desire a machine which can handle on the order of a hundred programs at a time on a sampled basis, out of a total program assembly of hundreds of programs. Each program would periodically receive a sample update; an average sample rate of about 50 samples per second per program would probably be adequate. This means that some 5,000 samples, or jobs, would be executed every second. The overall bit transfer rate for common memory, input-output, and messages is estimated as follows. An average of 25 words must be brought from common memory and 25 words stored there per job. This number is based on experience with the executive program structure of the Apollo Guidance Computer. Assume 50 bits per word for address and data. Assume an average of one input and one output message and four job assignment messages of 50 bits each per job. The minimum bit rate which could possibly serve this system is

$$5000 \frac{\text{jobs}}{\text{sec}} \times \left(50 \frac{\text{words}}{\text{job}} + 6 \frac{\text{messages}}{\text{job}}\right) \times 50 \frac{\text{bits}}{\text{word or message}} = 14 \text{ megabits/sec}$$

This rate takes no account of delays occasioned by stacked up requests or other access times, but is well within reach of today's technology for memory and transmission systems.

The instruction execution rate is estimated by assuming an average number, again borrowing from Apollo experience, of the order of a thousand instructions executed per job, and an average job duration of a millisecond. The latter figure is chosen on the basis of wanting the multiprocessor to react to an input event or job request within that space of time. This yields a figure of one microsecond per average instruction, and also implies that at least five processors need to be on line to handle the 5,000 jobs per second. Both of these figures seem extremely reasonable in the light of our expectations of the technologies involved; indeed, we expect that the technologies will soon substantially surpass these levels. This, added to the fact that we have been describing a somewhat primitive form of system organization, suggests that we may expect to have more powerful spacecraft data processors in a decade than there are on the ground today.

EXECUTIVE PROGRAM CONTROL FOR SPACEBORNE MULTIPROCESSING

ROBERT A. HOKOM

Mr. Hokom is a Senior Computing Engineer with the Computer Systems Group, Navigation Systems Division, Autonetics, a division of North American Aviation. Mr. Hokom was awarded a B. A. degree in Mathematics from the University of Southern California in 1959.

Mr. Hokom has been with NAA's Autonetics Division since 1962, working primarily in engineering application and systems programming. He is currently investigating software techniques for a NASA spaceborne multiprocessing study and is also directing a team in the development of a problem-oriented language and compiler for the on-board Minuteman computer.

EXECUTIVE PROGRAM CONTROL FOR SPACEBORNE MULTIPROCESSORS

By: Robert A. Hokom, Senior Engineer, Computing
Autonetics, A Division of North American Aviation, Inc.,
Anaheim, California

SUMMARY

The necessity for executive program control, a description of executive functions, and executive software implementation is discussed. The frame of reference is a spaceborne computer for a manned Mars mission.

The reasons for having executive program control include requirements for reconfiguration to handle failures, computational loading, and unanticipated processing as well as the general questions of efficiency and timing.

The executive functions described are program scheduling, inter-program communication, reconfiguration control, request processing, Input/Output supervision, and computer self-test.

The application of these functions for a multimodule computer, which is representative of many multiprocessors, illustrates their characteristics. Differences in executive implementation for other configurations are briefly discussed.

Although this paper is concerned with a particular mission for a single NASA study, the approaches and problems examined are applicable to a broad range of computer systems.

INTRODUCTION

The combination of a new class of space mission requirements and hardware concepts new to onboard computer systems requires a reorientation of software techniques.

Software design for current ICBM, manned spacecraft and unmanned space probe computers is primarily concerned with maximizing utilization within a framework of strict timing constraints. Reconfiguration, if planned for at all, is limited to switching in a backup for handling failures and to setting logic to enter alternate program paths for changing mission phases.

For long-duration manned space missions the emphasis on optimum utilization is balanced by the need for flexibility. The dynamic mission environment requires computer response, both hardware and software, to multiple failures, widely varying computational loads, and unanticipated requirements. Although timing constraints still exist, they are not as severe and a large portion of the computations are unconstrained.

A manned Mars mission (1980 time frame) is used in this paper as representative of this class of missions. After demonstrating the necessity for executive control in general, the various executive functions are described. Finally, an example of executive software for a particular configuration and a discussion of application to other configurations are presented.

NECESSITY FOR EXECUTIVE PROGRAM CONTROL

The Computations

Mars Mission Characteristics — The manned Mars mission used as the framework for this study consists of the following major flight segments:

1. Launch and injection into Earth orbit.
2. Escape and trans-Mars coast with midcourse corrections.
3. Injection into Mars orbit.
4. Escape and trans-Earth coast with midcourse corrections.
5. Earth re-entry and recovery.

The purpose of the mission is to perform extensive scientific measurements and experimentation in the Mars area, and a limited amount of the same during coast periods. It is representative of other long-duration manned space flights, such as orbiting stations, Mars landing, and other planetary missions, since their operations are computationally related.

Computational Characteristics — There are two distinct types of computations: control/monitoring and batch data processing. The first is necessary for guidance/navigation, status monitoring and communications, the second for scientific data handling.

Computationally, the mission consists of a sequential series of 20 unique phases which are classified as follows:

1. Mars orbital phase — Characterized by the mission's peak loading of 30000 words with 380,000 operations/second speed, and 0.5 hour recovery requirements.
2. Non-Mars, critical phases — loading of less than 12000 words with less than 200,000 operations/second, and 5 second recovery.
3. Non-Mars, non-critical phases — Loading of less than 24000 words with 250,000 operations/second, and 0.5 hour recovery.

The loading estimates include certain functions that are non-continuous. For instance, during the trans-Mars and the trans-Earth coast phases over half of the load is executed on command only at relatively infrequent intervals. Of course, other functions, such as

status monitoring and attitude determination, are continuously performed during the entire mission, although the mechanizations may differ from phase to phase.

Computational Requirements — The computer system must be flexible enough to functionally configure itself to efficiently process a variety of computational loads. This includes the capability for processing unanticipated programs (ie. existent program in-flight not scheduled for the current phase or the development of new programs through programming).

Certain portions of many functions must be executed periodically. The frequency of these cycles must not vary.

In addition to providing necessary error detection and fault isolation, the system must have a means for smooth transition to backup configurations.

Since in-flight programming might be necessary, software development must not be hindered by an excessive number of restrictions.

Dynamic Versus Static Approach

Static Programs — In general, given enough time, a group of skilled programmers can mechanize any reasonable set of requirements for a given computer without any special support software. In this case, in order to keep the computer size within reason, it is necessary to program phase defined "load modules," which are permanently stored on a mass storage device. Therefore, at least some means for sequencing load modules into the computer as new mission phases occur must be implemented.

These load modules could be mechanized as separate static programs, each with a high degree of optimality. This would mean that 20 unique load modules of between 12,000 and 30,000 words each would have to reside on the mass storage.

In order to reconfigure for a single failure, another complete set of load modules would be required. The handling of multiple failures would require multiple load modules for each phase. Reconfiguration to handle unanticipated processing would involve in-flight construction of an entire load-module.

Assuming that at least 100 unique load modules would be required, the mass storage would have to hold approximately 1,200,000 words and the logic to select them would not be insignificant. In addition, the thought of recoding functions on the order of 100 times is especially repungent to the programmer.

Dynamic Programs — With appropriate executive program control, each function can be mechanized just once and the load modules can be constructed during the flight from "load profile" tables. The mass storage would have to accommodate only about 60,000 words.

These programs will have to be mechanized so that inter-program data flow, timing, and Input/Output is performed with support software, and each program must be otherwise self-contained. There is some loss in utilization since there is executive overhead and modularization by nature implies some inefficiency, but the inherent flexibility is essential to achieving true effectiveness of the computer system.

An area often overlooked in computer system design is the effect of program modifications. As requirements are altered, deleted, or added, the associated programs must be reworked. Programs coded statically must be completely redone for each modification. Dynamic programming permits rework to be confined to directly affected functions.

Accepting the dynamic approach means that certain executive and support facilities must be provided, and that programming procedures must be established.

EXECUTIVE FUNCTIONS

Program Sequencing

Periodic Programs — There are a number of processes within each computational program that must be repeatedly executed at a fixed frequency. For the mission under consideration, the highest rate function is attitude determination which must be executed 20 times per second. Almost all others must be cycled once per second.

Background Programs — Other processes are to be executed continuously during a phase, but have no timing constraints. An example of this is status monitoring and testing during trans-Mars coast of vehicle systems to be used in Mars orbit.

Request Programs — There are many processes that are executed on command only, in other words they are non-continuous. An example of this is the guidance and navigation function during coast phases; it will be executed about once a day and will take only 0.5 hour. In terms of sequencing, unanticipated programs are considered as request programs.

Program Scheduling Routine — The executive must have a routine for scheduling the execution of these programs. The scheme must satisfy the strict frequency requirement of all periodic programs, sequence the background programs, and permit execution of request programs with some priority control.

This routine must minimize dead time and be able to perform its job for any of the load modules. This can only be done if the load profiles include information concerning the classification of each program in the load module so that scheduling tables can be constructed and updated.

Inter-Program Communication

Even though the various programs are functionally independent, there are a number of parameters that can be considered as "global." These include data referenced by more than one program, interface data for separate programs that jointly represent a single function, executive-computational programs' control and information parameters, and data necessary for initialization of programs.

There must be a means for global parameters to be transmitted to and from any configuration of computational programs.

1. Mars orbital phase — All connections are unblocked except that P_1 is able to temporarily block the P_2 - M_1 and P_2 - I/O_1 connections, and P_2 can do the same to the P_1 - M_2 and P_1 - I/O_2 connections.

2. Non-Mars, critical phases — A "processing group" consisting of P_1, M_1, and I/O_1 is formed by blocking the connections P_2 - M_1, P_2 - I/O_1, P_1 - M_2, P_1 - I/O_2, and P_2 - M_3. The P_1 - M_3 connection is also blocked but may be unblocked in order to expand the processing group's memory capacity. The corollary processing group P_2, M_2, and I/O_2, which is formed by the same blockings, is used as a backup to the other in case of failure.

3. Non-Mars, non-critical phases — The processing group of P_1, M_1, and I/O_1 is formed and the other modules are turned off. The M_3 memory module is brought into the group when required.

Multi-Module Executive and Support Software

Program Scheduling — The key area is the scheduling of the periodic programs. A requirement is established that all frequencies must be integer multiples of faster rates. This permits the usage of a fixed "time interval" cycle, which equals the highest frequency (0.05 seconds), as the base for scheduling. An interrupt system returns control to the schedule at this frequency.

The highest rate program is always executed first and then the next highest. A Periodic Table is maintained which contains frequency and status information. Figure 2 shows the execution sequence of a set of periodic programs.

PERIODIC PROGRAMS	FREQUENCY	TIME-TO-PERFORM
a	0.05	0.01
b	0.1	0.02
c	0.1	0.02
d	0.3	0.01

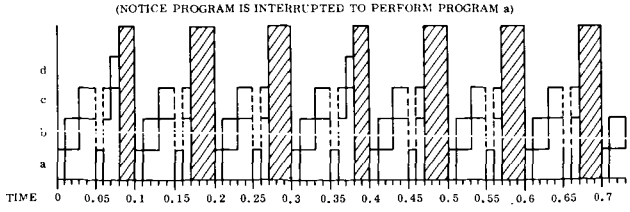


Figure 2. Example of Periodic Program Scheduling

The shaded areas in Figure 2 represent time periods when either request or background programs may be executed. A Request Queue, which contains requested programs and is ordered by priority, is exhausted before the Background Table is used to cycle the background programs.

During the Mars orbital phase a separate Periodic Table is assigned for each processing group. When the interrupt occurs, each processor scans its schedule. However, a single Request Queue, residing in M_3, is accessed by both. Separate Background Tables are used, though, when all requests have been satisfied.

When a program is interrupted, its register values and the program location counter is saved. A status indicator for each program is kept in the tables so that resumption can be scheduled.

Inter-Program Communication — Global parameters are assigned fixed locations. The various programs merely perform fetch and store operations directly.

Reconfiguration Program — Each load module is organized so that periodic, request, and background programs are separated into blocks. The memory is organized as follows:

1. Central data area and Executive programs.
2. Executive tables.
3. Periodic programs.
4. Background programs.
5. Request programs.

For the Mars orbital phase M_1 and M_2 would each be organized this way and M_3 would contain additional request programs.

To load a new load-module for a new mission phase, the current periodic block is moved to upper memory, overlaying the request block. The loader is executed as a TOP priority request, so this move is safe. The new periodic block, when completely loaded, is initiated; the old one has continued operating until this time. When the new periodic programs are operating smoothly, the other blocks are loaded.

A cold restart is accomplished by direct load. This is required if failure causes computer shutdown.

Unanticipated request programs are brought in by overlaying lower priority request programs.

Request Processor — Requests are made by setting logic flags in a Request Board that contains pointers to all programs. Console messages are interpreted by a console message processor routine to set this board.

The scheduler will check this board for possible additions to the Request Queue each time it is ready to handle request execution.

I/O Supervisor — Two special routines, PUT and GET, are used by the computational programs to perform I/O. The logical name of the sensor, which is supplied by the user, is used to scan a I/O Configuration Status Board for device selection. This board is updated by console inputs whenever manual I/O reconfiguration occurs.

Reconfiguration Control

An executive routine must be provided which can, in response to appropriate commands, produce a load module containing all currently required programs. This naturally leads to the requirement that all programs be relocatable so that memory resources are efficiently utilized. The scheme employed must be capable of reconfiguring in response to several situations.

Mission Phasing — When a new mission phase is to begin, the associated load module must be loaded in such a manner that a smooth transition of the computations and outputs is obtained. This is especially important for periodic calculations that will continue in the new phase.

Failure — The load module must be reloaded into either an equivalent backup or reduced computer system after fault isolation determines what errors have occurred.

Unanticipated Programs — When an unanticipated program is requested for execution, the current load module must be altered to accommodate the new program and, when necessary, the computer configuration must be expanded.

In all cases, the job not only is reassignment of resources (storage and processing time), but involves alteration of the executive itself so that adequate control can continue to be exercised.

Request Processing

Request programs can only be placed in the scheduling tables when a specific command is issued. There are two primary sources for request commands.

Programmed Requests — A program may wish some other program to begin execution when certain conditions are detected. The decision to issue the request can be dependent on parameter values, input quantities, or program sequencing logic. The priority of the request can be either fixed or a computed value.

Console Requests — The astronaut may want to initiate a request via the computer console. This can be accomplished by having a support package that will interrogate and interpret console messages and issue the appropriate request commands.

A routine to accept these requests and set up the appropriate logic in the scheduling scheme must be available.

Input/Output Supervision

The functional configuration of the Input/Output (I/O) system, which includes computer I/O units, sensors, and their interconnections, is dynamic also. Thus, reconfiguration is concerned with this area too.

Since the recovery requirement during 90 + % of the mission is 0.5 hour, the main means for accomplishing reconfiguration will be manual replugging of sensor lines into I/O conditioners. During critical phases the critical I/O devices will be redundantly available and automatic switching can be used well within the 5-second limit.

The computational programs could not keep up with these changes. Therefore, a central I/O supervision routine must be provided. Input/Output will be performed by appropriate calls on this supervisor from the various computational programs. The current status of the I/O system must be updated via the console when manual alterations are performed.

Self-Test

Whatever error detection and fault isolation techniques are employed, the software to support it comprises another executive function.

Application to Specific Computers

The logical design of the computer system will have a direct bearing on the relative difficulty and importance of implementing these executive functions. In fact, the hardware design must be concerned with providing features that will facilitate the executive software system. Thus, the total computer system design effort involves considerable feedback and trade off evaluation between the hardware and software areas.

A REPRESENTATIVE SYSTEM

Multi-Module Computer Description

The software design for a multi-module computer capable of performing the manned Mars mission is demonstrative of an executive implementation.

Hardware — The computer is composed of a number of modularized components as represented in Figure 1. There are two processor modules (P_1 and P_2), three memory modules (M_1, M_2, and M_3), two I/O modules (I/O_1 and I/O_2), and a variable number of I/O conditioners with attached sensors. As can be seen, there are memory-processor connections, processor-I/O connections, and I/O-conditioner-sensor attachments.

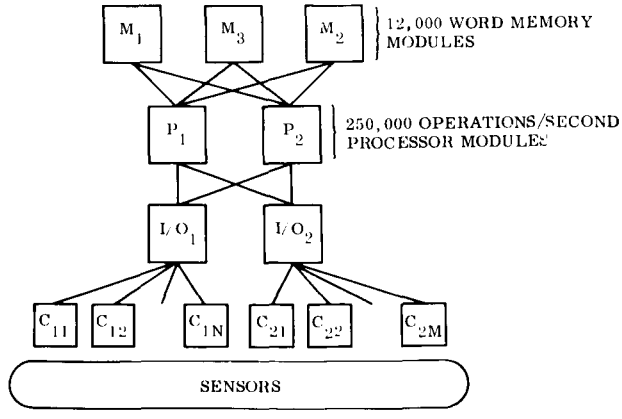


Figure 1. Multi-Module Computer Hardware Representation

Functional Configurations — The functional configuration is changed by blocking and unblocking interconnections depending upon the type of phase being executed; these are:

On input the sensor is sampled, an appropriate reasonableness test may be used to check the data, and the value is passed to the user. On output the value is picked up, transmitted, and verified via hardware feedback.

Self-Test Program — For this system a number of error detection techniques are employed. Fault isolation, except for I/O errors, is performed after failure has caused a backup configuration (another processing group) to be brought up. The means used are the following:

1. Pulse stream — A special flip-flop, which will be hardware monitored, is complemented at the beginning of each time-interval cycle within the scheduler. Processor logic errors, interrupt errors, and some memory failures are detected.

   Failure notification will automatically be given when the pulse stream test is failed. Errors detected by other means perform notification by forcing pulse stream failure.

2. Arithmetic unit test — one of the periodic programs is a routine that performs a complete check of the processor's arithmetic logic.

3. Check-sum — The computational programs are internally organized so that code and constants are blocked separately from variables. Check-sum information is contained in the load profiles and is kept in the scheduling tables. When the scheduler selects a program for execution, it first performs a check-sum on it.

4. I/O tests — Another periodic routine issues special test parameters through I/O conditioners and checks built-in feedbacks. The PUT and GET tests are also used to detect I/O errors.

   A Conditioner Status Table is used to collect information on which I/O errors have been detected. This is used to isolate to an I/O unit, a conditioner, or a sensor. Only the first will cause computer failure notification.

Other Configurations

Multi-Computer — The software for a multi-computer system, which also is able to perform the mission, closely parallels the multi-module's. The primary difference is that during the Mars orbital phase each computer operates under completely separate executive control.

Distributed Logic — The distributed logic configuration considered best suited for the mission is one with a number of separate cell groups connected only by an Inter-Group Buss.

Task modules, which are blocks of code that fit in one cell group, are scheduled only for requests. Periodic scheduling is performed locally within the task modules. Reconfiguration consists of reassigning cell groups to new task modules. Errors are detected and isolated at the cell group level.

The inter-program communication function is vastly increased in scope. An Inter-Group Communication System is used to schedule the buss on a time-shared basis for transmission of inter-task data, global parameters, I/O values, and certain test indications.

Unexamined Systems — Although these three designs are fairly representative, there are innumerable computer designs that might be considered for spaceborne missions. The executive functions that were discussed must be incorporated, to some degree, in any design.

ERROR CONTROL

SELF-REPAIR: FAULT DETECTION AND AUTOMATIC RECONFIGURATION

EARL C. JOSEPH

Earl C. Joseph is a Staff Scientist at UNIVAC Defense Systems Division (St. Paul, Minnesota), Sperry Rand Corporation. Mr. Joseph joined UNIVAC in 1951 after receiving a B. A. degree in Mathematics from the University of Minnesota.

Until 1955, he held positions of Mathematician, Programmer, and Applications Analyst in UNIVAC's Arlington, Virginia, 1101 Computation Center. Since then he has held various supervisory and managerial positions on the Nike-Zeus and Nike-X efforts. In this capacity, he managed the systems design, logic design, programming, and application of four generations of large-scale computer systems, including: (1) the Target Intercept Computer (TIC), (2) the General-Purpose Digital Computer (GPDC), (3) the MAR (Multifunction Array Radar Computer (a GPDC), combined with a centralized digital controller for distributing parallel control commands to the array radar, and (4) the Nike-X Multiprocessor.

In his present position as a Staff Scientist, to which he was appointed in 1963, he performs advanced systems design research for ground base and aerospace computers.

 N 67-17106

SELF-REPAIR: FAULT DETECTION AND AUTOMATIC RECONFIGURATION

By Earl C. Joseph

UNIVAC DEFENSE SYSTEMS DIVISION
SPERRY RAND CORPORATION

SUMMARY

The reliability and flexibility of next generation spaceborne computers will not be gained primarily from the application of new and more reliable electronic devices, but rather through system organization to meet the reliability requirements of the space age. The multiprocessor is an example of such a computer organization. Multiprocessors are capable of parallel processing and can be configured and reconfigured for general-purpose applications meeting advanced requirements for reliability and adaptability.

With the advent of large scale integrated circuits (LSI) we are told that we soon will have "computers on a chip". With such LSI chips containing full or partial systems, it becomes practical and economical to implement self-repair. That is, the addition of spare redundancy and diagnostic logic on a chip is possible without materially increasing the cost and size of the system. In addition, by including the "spares" on the chip ready to be "fused" into usage, automatically under program control, self-repair is accomplished without additional connections and complex switching logic.

Included in this paper is a description of the fault detection, isolation, and location techniques required to recognize a system failure, to make the necessary real-time self-repair adjustments to the hardware configuration, and recover from errors generated. This multiprocessor organization with nonmanual self-repair features allow for reconfiguration so that a level of capability is continuously maintained and 100 percent systems availability can be virtually assured.

INTRODUCTION

Next generation computers will be required to have available a maximum capability 100 percent of the time for many applications.

With the advent of modular multiprocessors[1], the design of systems capable of achieving self-repair is possible so that 100 percent availability becomes possible.

If parts fail the redundancy of a multiprocessor organization can be used to obtain reliable operation, that is, a multiprocessor is inherently a space redundant system.

Increasing the scope of applications of Large Scale Integrated Circuits (LSI) is simply a matter of time; for its usage as the principal ingredient in spaceborne and aerospace computers (and for that matter, any computer) is a certainty in this decade. The first computers that will be made completely from LSI will be for Aerospace applications. Of course, the major incentives leading to the incorporation of LSI, is the promise of substantial system cost reduction and a considerable increase in reliability.

The ushering in of LSI puts us at the threshold of fourth generation computers. Like previous generations, the current one is characterized by a dramatic breakthrough in component/device technology. The four generations and their associated state-of-the-art technology are:

- 1st generation computers - Vacuum tubes (1950-1957)
- 2nd generation computers - Transistors (1956-1966)
- 3rd generation computers - Integrated Circuits (1962-197?)
- 4th generation computers - LSI (1967-?)

The continuing reductions in size, cost, and power consumption of logic circuit elements through the use of LSI encourages and facilitates the utilization of more complex logic networks in digital computers for spaceborne applications and allows for practical self-repair. This LSI-provoked revolution occurring in the electronics industry is drastically changing computer components and is causing an upheaval touching all levels of space technology.

Before LSI, the designer was forced to be concerned about the amount of logic going into the make-up of the spaceborne computers. In the near future, that will not be the case; for doubling or tripling the amount of logic per system on a wafer makes only a small difference in cost, size, and power consumption. Thus, a new era of highly capable and extremely reliable computers can now be considered by the space planner.

Historically in the computer industry great advances have been made in designing for reliability. In 1951 the early computers had a mean-time-between-failures (MTBF) which was less than one hour. Today we are using a few computers which exhibit an MTBF of thousands of hours and are designing computers which should have an MTBF of 10,000 hours or more. This paper describes design methods to increase the reliability by another order of magnitude, to the range of 100,000 hours MTBF or more.

LSI means more logic per component to the system designer. Projections made at the recent (1966) IEEE Lake Arrowhead Workshop, "The Impact of Large Scale Integration on Information Processing Systems", by LSI component manufacturers indicate that one can expect hundreds and thousands of logic gates per component and in the not too far distant future, in the 1970's, it will be possible to design computers using ten thousand gates per component. Computers built today use integrated circuit components with two to four or at most ten gates per component. So even with a few hundred gates each, an order of magnitude breakthrough is occurring and the future promises breakthroughs of far greater magnitude. These revolutionary changes mean higher speed and smaller future systems and are of such magnitude that a revolution is occurring in Aerospace computer design.

Since the component size is relatively the same size throughout this revolutionary change higher speeds and smaller systems are possible. This results from smaller and closer gates driving shorter wires. For supporting long-term missions demanding increased data processing capability, these features inherent with LSI, mean that the computer needs of future high-capability, post-Apollo, spacecraft can be met. Of equal and perhaps greater importance to the spacecraft is that these LSI features also require less power than present day circuits and systems.

On-board spacecraft reliability requirements pose a formidable problem to the computer designer of yesterday. With LSI, where orders of magnitude fewer individual components, connections, and process steps are required to implement a computer, orders of magnitude greater reliability can be achieved. This improvement in component reliability coupled with a modular multiprocessor organization capable of reconfiguring itself and self-repair to accommodate both equipment failure and mission changes will allow orders of magnitude improvements in computer systems reliability and availability.

In order to meet the spaceborne requirements of: 1) a self-repairing system capable of diagnosing itself, 2) a spaceborne computer system capable of simultaneously performing a wide variety of either or both command and control or mission data processing tasks, and 3) a system capable of reconfiguring itself into a functional system,
a multiprocessor computer system organization is optimum[2].

Self-Repair: A Definition

A self-repairable digital computer is a reliable automaton which has the capability of automatically detecting and isolating a failure to a functional subsystem, then automatically causing a program (or hardware) to switch a spare functioning subsystem into the system to replace and repair the failure. This paper describes methods of designing computers for continuous operation through self-repair where manual repair is not possible.

System Reliability and Self-Repair

Obviously if the total computer system is demolished, it is not capable of self-repair. What then is the minimum subsystem configuration which must function to allow the computer system to be self-repairable? This paper describes a self-repairable system requiring a minimum of operating parts of the system together with spare submodules that can be switched into the system to replace failing subsystems.

The reliability of a group of modules in series, presents a serious problem in maintaining the system in operation; for, the failure of one module will disable the entire complex. The solution is to use parallel redundant units (as in a multiprocessor) and interchangeable standby spares which can replace failing modules.

Because the multiprocessor is made up of many modules that perform each function, it is less vulnerable, in military applications, than contemporary unit computer organizations; for if one or more modules are destroyed, by one means or another, and a module of each function still exists, the multiprocessor can still accomplish its job. Thus, future computers for military applications will be organized with parallel operating functions like a multiprocessor. To further minimize the vulnerability of the computer, the system designer has two choices: 1) distribution of the functional modules throughout the physical system; or 2) centralizing the functional modules into a well protected area.

For several years UNIVAC has examined various computer configurations in a continuing search for a more reliable computer system. These studies conclude that a modular system, which can adapt itself to the specific task, is needed. A system which can recognize the failure of a functional unit and take corrective action is essential. Future applications require a system design which allows not only graceful degradation at the module level, but which also permits adjusting the tasks to be performed to the remaining hardware capabilities. That is, as errors occur, software recovery techniques must enable and assist the system in recovering from malfunctions. At the full operational level, all of the hardware consisting of many similar functional modules in a multiprocessor is needed and used in a nonredundant manner. As a functional unit fails, it is electronically removed from the system, new data paths are created by switching, and the system continues at a temporarily reduced capability or capacity. With the advent of multifunction integrated circuits, manufactured using batch processing[3] consisting of hundreds and thousands of logic functions per component, this type of automatic electronic self-repair becomes feasible. The functional units are selected (during design) in such a manner that it is statistically improbable for a combination of failures to reduce the capability below the predetermined level needed for minimal system operation. The hardware and software diagnostic system must further locate the fault in the failed functional unit to a replaceable unit without interfering with the system operation. The failed unit can then be automatically repaired and the system returned to full operational capability.

System organization techniques such as these enable the computer system to achieve system availability and reliability several orders of magnitude greater than that of the functional units involved. Summarized, this concept is one of graceful degradation down to a predetermined activity level. The probability of failures which reduce the system to a lower level is so small that it is realistic to guarantee operation at this predetermined level. Thus, extreme system reliability can be achieved through system organization and design rather than from circuit improvements only. In the past, achieving a system reliability greater than the reliability of the components was only possible in the neuronal systems encountered in biological systems. With the advent of parallel systems like the multiprocessor, however, man can now achieve a similar level of reliability in the hardware systems he builds.

Any computer system at some point in time will have subsystems that fail. By advance planning and using a multiprocessor organization, the system designer can design a computer system to perform its task at, perhaps, reduced speed even while containing failed subsystems. Both the hardware and software,

however, must be designed to work together to achieve total system reliability and continuous availability.

Today's individual solid state components are extremely reliable, in the neighborhood of one failure per 10 billion component hours. Since it is prohibitively costly to obtain sufficient information about failure mechanisms to improve component reliability beyond this point, there is little likelihood that the components of the future will be more reliable. Today's component, however, represents one or only a few logic functions, whereas tomorrow's multifunction, batch fabricated, LSI circuits will contain hundreds, thousands, and even tens of thousands of logic functions. In this fashion, with the reliability of the component no better than today's, the actual reliability of each logic node will be increased by one, two, three, and even four orders of magnitude.

Further, since there is no observable or predictable deterioration with time, associated with solid state components, the system designer of a self-repair system using replacement modules, need not be concerned with routine component replacement schedules. That is, there exists no algorithms to tell the designer when a component is about to fail because there are no known wearout mechanisms, other than random failures. Thus, the designer is faced with designing the self-repair system to replace failures only after a failure occurs. In addition, redundant systems gain no added reliability from standby modules over active redundant modules. If the redundant modules are active, an additional capability is achieved. This capability gain can be used in a fashion to allow the system to be smaller, when designed as a multiprocessor, than a system with an inactive standby and to achieve more reliability by using fewer components in the total system.

SELF-DIAGNOSTICS

In a computer system consisting of many computers, such as a multiprocessor, it is possible to design the system.to permit a functioning processor to diagnose and repair other parts of the failing system.

For example, consider a worst-case situation where all diagnostic aids fail to isolate the failed subsystem. Then, under program control, a functioning processor need only, by trial and error, switch in and out functioning subsystems in the failing system until a functioning and repaired computer is achieved.

Briefly, the techniques of self-diagnostics, to be discussed, assume that a processor has access to the various internal or controlling registers of the other modules in the total system. Thus allowing a processor to actively diagnose faults in itself and other subunits in the system in real time. In this manner, a processor can actively detect and control errors as they occur and reassign tasks within and among these devices to compensate and adjust the systems work load among the remaining modules until the failure is repaired.

During the latter part of the last decade and continuing through to the present time, UNIVAC has been active in the development of software[4] which has the capability of recovering from transient computer and system errors. Executive control and error control routines were developed which, in conjunction with hardware aids, recognized large classes of computer and system errors. When errors were detected, the control was transferred to the appropriate self-analysis routine so that appropriate emergency corrective action could be taken. In demonstrated cases, useful results were produced even with several errors occurring per second.

With the arrival of LSI circuits containing many logic nodes and even complete functions, component reliability has and will be greatly enhanced. The demands upon reliability at the system level, however, have become so great that mere attention to circuit design, component selection, and manufacturing techniques will no longer suffice.

In designing computer systems with self-repair features for extreme reliability, the designer needs to consider many things, many more than one can possibly discuss in a paper of this scope. Some old assumptions, definitions, and new considerations presented without proof are:

- Failures are malfunctioning hardware or software that may or may not cause an erroneous calculation. They may be either intermittent or catastrophic. Whereas errors always result in an erroneous calculation. Errors are caused by either malfunctioning hardware or partially debugged programs; either type of malfunction may or may not generate an error; for example, consider a component failure in the multiply algorithm: if the multiply instruction is not called upon and thus not executed, no error will exist. In the design of a self-repairable system it is not just good enough to repair malfunctioning hardware; the system must also be designed, as described herein, to recover from all errors, whether caused by intermittents or catastrophic failure. The method of error recovery to be described uses both hardware and software, working together, to self-repair the damage caused by an erroneous calculation. Such damage is manifested as unfinished calculations, loss of inputs that require reconstitution, and so forth. Malfunctions which do not generate errors are not detected until they cause an error.

- Errors and failures occur infrequently in todays debugged systems using highly reliable components; therefore, a computer system should be designed so that little or no extra system time (additional time to perform calculations) is required for error detection and error control during normal operation. That is, the computer can be designed to take extra time (and use additional logic) on an emergency basis at the time errors occur.

- In a self-repair system, spares that would otherwise be lying on a shelf until they are manually inserted, can be utilized for automatic replacement. The number of spare modules that can be handled automatically, however, is small because the switching logic increases rapidly as the number of modules switched increases. In a practical self-repairing system, switchable spares

must contain a lot of logic in order to keep the number to be switched small.

- If error control is preplanned such that the programs pre-condition the system for error control, it is possible to recover and self-repair the damage resulting from errors. (Refer to section on adaptive error control.)

- The number of connections between modules is a minimum when a total function is included in the module. That is, the density of connections within a function is high whereas the density of connections between complete functions is low. Thus for practical self-repair a functional break up of the logic is desirable to keep the number of lines to be switched at a minimum.

- Assuming that the individual components in a system offer maximum reliability, it is a matter of system organization to achieve a greater system reliability than the individual components. In general, to achieve this higher reliability the designer must use one form or another of redundancy; however, a multiprocessor is already a redundant system by definition. This paper describes a method of achieving extreme reliability by making use of the redundancy of a multiprocessor and by using a computers spares without resorting to total system triplication. The method to be described uses many techniques; fault-masking hardware networks where they are required, auxiliary coding detection schemes (parity and the like), and both software and hardware aided detection and correction schemes together with self-repair by spare replacement. In general triplicated faultproof[5] combinational networks are not used for the sake of economy. Such a system combining methods of fault masking and replacement strategies for achieving reliability, using the method best suited to the case at hand, leads to a more economical and more reliable system - a system having the widest range of tolerance.

The most reliable computer in the world would be unreliable and useless if the programs it is executing were not completely debugged and designed for reliable operation - designed to recover from errors. To this end, an adaptive error control program which adapts to a malfunctioning environment is required.

In order for a computer to repair itself and recover from errors the following steps must be accomplished:

- Error Detection
- Fault Location
- Fault Isolation
- Error Control and Recovery
- Repair Replacement.

The following tabulation is a sample list of the type of error detection circuitry that may be included in a self-repairing computer system:

- Parity checking for data words
- Parity checking for memory address words
- Parity checking at functional unit interfaces
- Integrity checker on the program address counter
- Over-write and over-read checkers for list memories
- Illegal operation detectors
- Arithmetic error detectors
- Power and temperature (environment) fault detection
- Real-time checks
- Critical command operation checks.

In addition to the circuitry required to detect the above failures, there are fault registers which freeze information about errors as they occur together with associated fault interrupt generation control.

Errors and failures in computations are detected by either hardware or by programmed tests. In a system which incorporates considerable error checking hardware, all programs can become fault checking programs. Such a system would consist of some or all of the following error checking hardware:

- Data parity checking - all register-to-register transfers and all data read from memory.
- Address parity checking - all transfers of addresses between registers and read or write references to memory up to the input of the memory drivers. It is far more undesirable to jump to a wrong address, read the wrong word, or write into a wrong memory location than to read a data word that has lost a bit. Thus, this type of parity check is more important than the data parity check found in most computers. Yet, it is amazing that address parity checking is seldom implemented in contemporary computers.
- Program address counter parity check - before advancing the counter, the parity of its value plus one is predicted and then, after advancement, its parity is checked against the predicted value. Again, this is an extremely important check, because if the program address counter is not operating properly, no believable program execution is possible. Operations like this are termed critical commands. Critical commands are defined as that logic which, when not functioning, does not allow programs to be executed.
- Sequence time checking - all instructions preload a countdown timer before execution and, if the timer reaches zero before the instruction is completed, a hang-up error exists. An error interrupt occurs to release the computer from this condition.
- Arithmetic checks - checking for overflow and the like.
- Illegal operation checks - such as nonexistent address checks.

- Environmental checks - examining power and temperature. To further reduce the degree of system exposure to failures, redundant nodes and modules are powered from individual and separate power sources which have individual turn on/off systems, controlled by the power checking system, so that one of the nodes is always operable.
- Memory lockouts - controlled by the executive program for protecting data and programs being executed concurrently.

FAULT RECOVERY

The main objective of a self-repairable system is to maintain the system at its maximum possible capability. This goal is accomplished by detecting a failure as soon as possible after it occurs, isolating the failure to a functional subunit before system contamination can occur, diagnosing the failure to the replaceable unit, replacing the failing unit, and re-establishing the function within the system. To accomplish the above operations in real time all operations must be manipulated and controlled by the programs and hardware without manual intervention.

Another objective of such a system is to maintain a continuous level of computer capability within the system. This means that the removal and replacement of a faulty unit should not interfere with the operational capability of the remainder of the system. These objectives are not singularly limited to the computer, but rather include the total system: the computer, its software, and peripherals. The self-repairing system discussed herein is for the total system.

In general, errors will be detected by hardware and software. Detected errors will then be recorded in a processor, a memory, or a status unit. The error detected will react in the system with the initiation of a task through an interrupt at one of the functioning processors. The processor interrupted will analyze the trouble, determine corrective action, and decide whether to continue, do further diagnosis, or isolate and replace the troubled component. In most cases, the program will decide how many times an error may occur before isolation is necessary. Electronic isolation and surgery will be accomplished by changing the status of the defective unit at a status unit. Functional isolation and system reconfiguration are accomplished by programs, switching submodules, and using memory lockouts. Subsystem switching (replacement repair) will be done without power shutdown and confidence checks will be made before returning the electronically repaired equipment to operational status.

In all these cases, if an error is detected, the following sequence occurs:

- An interrupt occurs to the appropriate executive error control program which further resolves the error condition. This error control program, if it is executable, will determine if the error is intermittent or catastrophic. If a failure has occurred, it will indicate this condition and initiate repair. If the failure is such that the executive error control program is not executable, then special hardware or another processor is required to isolate the error to the functional subsection that has failed, and the error control hardware initiates the repair by initiating a functioning processor.
- Prior to an error interrupt and instantly upon detection of an error, information about the error is captured and frozen in registers. Two types of data are captured and saved: 1) the type of error and 2) an address associated with the error. This information is used by the error control program or hardware to determine which functional subsystem has failed and for recovery from the error.

Additionally, real-time control programs and certain on-line multiuser systems require that all outputs are checked to determine their validity and the checking of arithmetic operations. In either case, it is usually sufficient to execute periodically (for example, before a control output) a confidence reliability test program to determine the state of the system, and if no errors are detected, assume that the system was also all right when the critical computations were performed. This program also checks logic which does not have associated error checking hardware. In other cases, additional logic is required to check all operations.

To assist in the error detection, location, isolation, and recovery process, the following registers are required in each processor:

- Error Type Detected - Holds type of error
- Error Address - Holds address associated with error
- Error Recovery Location - Program pre-loaded for recovery (to be described)
- Memory Lockouts - Holds lockout information
- Status - Holds operatable status of all subsystems.

In contemporary computers using discrete components for achieving a logic node, parity checking is a useful method of detecting single errors occurring in transfers of data. Parity checking has found widespread use in the computer field as an economical method of checking for errors. With the advent of the LSI circuit containing many logic nodes, however, the probability that a multiple logic error occurring becomes very likely. Since there are many logic functions per integrated circuit and when a failure occurs, such as a crack propagating through the single component circuit, the probability of the failure affecting many of the logic functions, rather than just one of these functions, is high.

With the probability of multiple logic switching function failures occurring when a component fails, the possibility of an even-odd parity check detecting the error is considerably reduced over the days when discrete components were used in systems. This means the value of parity needs to be reanalyzed in the light of this new problem encountered through the use of batch fabricated integrated circuits. In the meantime, while other economical methods of detecting errors are determined, the designer is burdened with arranging his logic so that more than one logic failure in a single component does not invalidate the parity checks. As the number of logic nodes per component increases, the

number of external connections to the component for performing the logic functions goes down rapidly with a correspondent increase in reliability. That is, the number of connections between natural logic functions is considerably less dense than the connectivity required within the function (within the integrated circuit). When the logic designer designs around this parity problem, the number of connections to the integrated circuit goes up somewhat in order to isolate and achieve valid parity checking and, in turn, reliability goes down because of the extra connections, however, the increase in connections is less than those deleted through the use of multifunction integrated circuits.

To determine what portion of the diagnostics is to be performed by the hardware and what part by the software, the designer must determine, for the various applications of the equipment being designed, what period of time is tolerable for system interruption. In general, error detection must keep pace with the computations and thus hardware is required for error detection; whereas, fault location, isolation, replacement repair and process restoration need not keep pace with the computations and these functions can be performed primarily by the software. The question becomes one of determining how much hardware to put in the system for error diagnostics. This becomes an important question when the designer realizes that only a very small percentage of the total system will be required for a coupled error detection and self-repair scheme on up to three or four times as much hardware for a triplicated system. Studies to date indicate that replacement systems require a percentage increase in hardware rather than the many-factor increase required when massive redundancy is used.

PROGRAMMED SELF-REPAIR

The pool of submodules, a spare parts bank, is used by the self-repair program in reconstructing those portions of modules that have failed by switching, see Figure 1. In effect, the program switches out the failed module. The error control program then determines which submodule has failed in the failing module in order to switch it out and to switch in a good submodule from the pool of spare submodules. Total system capability is regained by switching this "repaired" module back into the system.

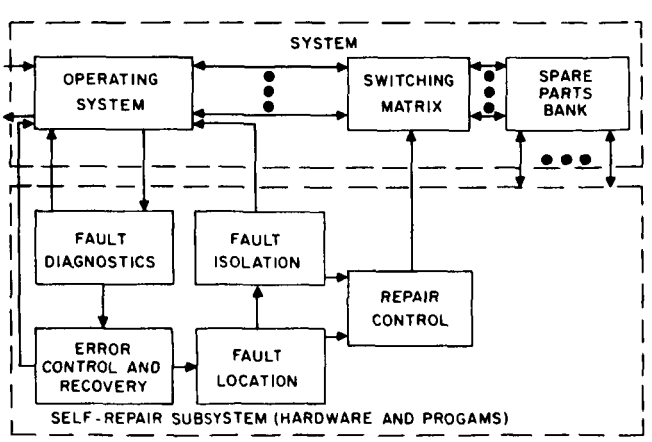


Figure 1. A Self-Repairing Machine

Critical programs, constants, and computed variables are double-stored in separate and distinct memory modules. In particular, the self-repair and error control programs are double stored. Thus, if a memory module fails, the alternate module containing identical data is referenced. If a processor (or input/output) module fails, a functioning module takes over the task of the failing module. In either case, the total task is performable, at a possibly reduced speed, until the failing module is repaired.

ADAPTIVE ERROR CONTROL

There are many methods for a program, which has been interrupted because of a fault, to recover (that is, to adapt to the error). One would be to return to some previous point in the program where all computational values are present and repeat a section of the program that has failed. Another method would be to use an acceptable substitute for the desired output value and continue the program without the information from the failing section. Other methods would be to perform entirely different calculations or reconstitute the inputs and reperform the computations. Each recovery method is unique to the program which has been error interrupted. The programmer of such a system must be able to select the best method of recovery for each critical portion of his program.

One method of achieving failsafe[6] adaptive error control is to preplan for the occurrence of errors. In this method the system is preconditioned during the execution of the program, as each critical subprogram is initiated, by logging a recovery address (where to recover to) into each processors fault recovery address register. After an error interrupt (detection of an error), the address in this register is used by the executive error control program to determine where to recover to, in the program interrupted. This method allows all faults, including transients, to be recovered from and corrected.

Examples of fault recovery initiation points are denoted in Figure 2 by the encircled letters A, C, and D. As the program reaches each of these recovery points, the location of the recovery point is sent (retained in memory) to the error control subroutine in the executive control program. Only one recovery point (address location) is retained per processor at a time. Whenever a fault is detected, control is transferred to the executive error-control program (usually through an interrupt via a task list), which uses the address thus retained (A, C, or D) to cause a jump to where the appropriate remedial action will be initiated. For example, assume an error occurs in the middle of Subtask 2 in Figure 2.

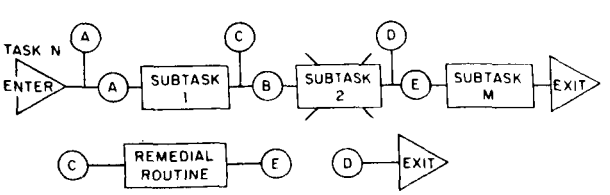


Figure 2. Adaptive Error Control and Recovery

Control is transferred to the executive error-control task, which, after initiating a diagnostic task, transfers control to subtask C (the fault recovery point) which performs the specified remedial task before any erroneous external effects occur.

By incorporating adaptive error control in this fashion, a processor becomes a self-recovery system – a time redundant system.

The following examples are some of the criteria for choosing the recovery points:

- A remedial routine must be performed
- Information must be read again
- The problem must be computed again
- A set of computations must be aborted
- Data must be reconstituted.

This error control philosophy is the result of studying programmed systems using this technique successfully. The following real example describes the kind of results possible with this method of error control. In one system built by UNIVAC, some calculations and all errors were event-recorded. During many runs involving a real-time control problem, a memory short developed which associated the computer electrically to other equipment in the metal shielded room. Consequently, an error occurred whenever a telephone relay clicked, whenever an electric typewriter was operated, and so forth. The event-recorded tapes indicated a mean-time-between-errors of 30 milliseconds. Even with errors occurring at this rate, the real-time control task was accomplished satisfactorily.

Thus, all programs, through adaptive error control and error detecting hardware become fault checking and correcting programs.

Briefly, the end goal of adaptive error control is not to achieve internal fault free operation but rather to achieve error free system operation as viewed from the outside.

RELIABILITY AND AVAILABILITY

The multiprocessor system described herein was simulated using pertinent calculated reliability parameters. The purpose of this simulation analysis was to show how levels of organizational redundancy, repair philosophy, and component reliability interact and affect the reliability of a self-repairing multiprocessor system.

All units in the system, were assumed to have exponentially distributed failure times. See Figure 3. The repair rate using a pool of submodules for replacement was assumed to be the same for all units in the system. It was also assumed that the system was to operate continuously for a one-year period without manual maintenance and upon failure was to be serviced automatically by the error control programs and self-repair system from a pool of spare submodules so that failing functions could be restored to the system; however, each submodule was lost from the system upon failure. The mean-time-to-restore a failed unit to service was assumed to be lengthy compared to the actual milliseconds required by the programs.

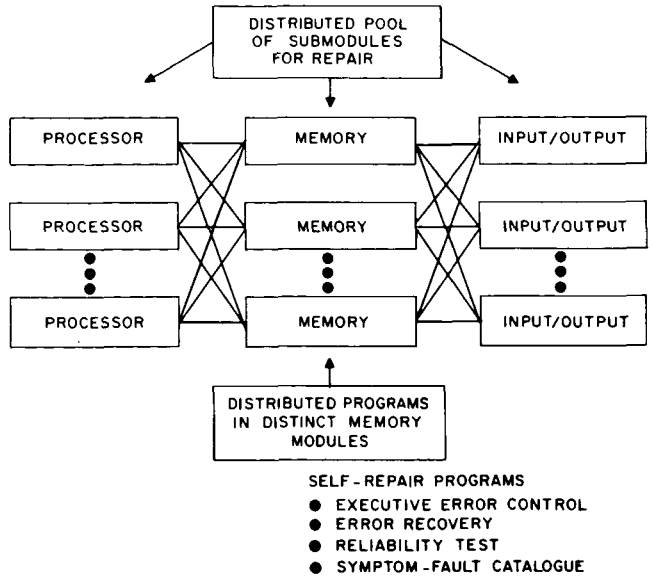


Figure 3. Self-Repairable Multiprocessor

The system was assumed to have failed when: 1) all modules of a function fail simultaneously (e.g., all processors); or 2) when all spare submodules of a type were exhausted.

Each processor was assumed to consist of 10,000 logic nodes, contained in 250 integrated circuit packages. A typical component failure rate for each package was selected as 25 failures per billion hours or in other words, a meantime-between-failures per component of about 5000 years.

The input/output units were assumed to contain as many packages of the same type as the processors. The memory modules were assumed to consist of less than half the number of packages as a processor. In all cases it was assumed that the switching logic and power were included in the modules. A two-processor, two input/output, three-memory module multiprocessor, with spares consisting of 2000 integrated circuit packages (80,000 logic nodes), was analyzed.

A pessimistic estimation for the mean-time-between-failures was calculated using the method of Kneale[7].

The result was a mean-time-between-failure of more than 100,000 years.

That is, a system consisting of 2000 packages using the methods outlined in this paper for obtaining reliability is 20 times more reliable than its individual components.

SPARE REPLACEMENT SWITCHING

A major problem in achieving practical self-repair is in performing the electronic surgery (the logic) for switching out a failed submodule and switching in a

functioning spare submodule (replacement repair). One method of solving this problem is to:

- Place spares on same LSI wafer as logic being spared in order to reduce the number of external connections.
- Use logic to perform an analogous operation of fusing for the interconnection of submodules including spares.
- Place the fuses in serial for each interconnection so that a malfunctioning submodule can be removed both functionally and electrically from the system - the blowing of these fuses deletes the submodule from the system. The reverse operation of the fuse (making the circuit) allows the spares, which are bussed in parallel to the interconnection path, to be added, both electrically and functionally, to the system.
- Allow a computer to self-repair itself when critical control logic fails. The logic required simply tries connecting and disconnecting (by fusing) submodules until a functioning set of submodules, to make up a complete system, is located.
- Use an error status register to address (determine) which interconnection lines and control logic to effect the fusing operation of switching the failure out and switching a good submodule in, when malfunctions occur. This register is loaded automatically on detection of critical control errors, which also automatically energizes the control logic for the fusing operation. For non-critical failures (failures which allow a program to be executed) a program is required for loading the register and initiating the fusing operation (for the purpose of economy of hardware).
- Use multiple fuses for spares that are usable in more than one section of the system. When a submodule is connected into the system for operation in one section, the other fuses are interlocked so that the same spare cannot be fused into a circuit for a different use in the system at a later time. This method of fused switching allows submodules together with their fuses to be added modularly.
- To each submodule there are multiple fused interconnection paths including power to provide redundant pathways. Therefore, even if multiple failures occur, the submodule can be switched out so that it does not affect system operation.

Since it is desirable to design computer systems using a minimum number of spares in order to keep the switching logic within practical limits, an investigation was performed using quorum probabilities. It was assumed, since it is not possible to predict in which submodule a failure would occur, that the self-repairing system would require at least one of each different type submodule as a spare. Further, in order to achieve the desired high level of reliability, additional spare submodules would be required. By using Einhorn's[8] equations for the calculation of MTBF and solving for the number of spares (redundant submodules) it can be shown that the number of spare submodules of each type is small.

UNSOLVED PROBLEMS

Not all problems for achieving self-repair have been solved. The problem areas associated with self-repair techniques requiring solution are:

- What self-repair techniques applied, singularly or in combination provide the greatest improvement in reliability?
- What methods are optimum for automating and initiating:
  - Fault diagnosis
  - Fault location
  - Fault isolation
  - Self-repair by replacement
  - Error repair (process restoration)?
- What constitutes a complete (closed) set of fault diagnostics and self-repair techniques and what theory can be formulated to show that the set is complete?
- What is the effect of self-repair on the total system relative to design, manufacturability, maintenance, application, etc. ?
- What ground rules must be followed to achieve total self-repair?
- What are the implications of self-repair on programming?
- What different diagnostic and self-repair techniques are required by various functional logic circuitry, such as control and timing logic, hard core logic, critical command logic, and memory?

CONCLUSION

With micro-miniaturization techniques growing into standard usage, that are difficult and time consuming to repair manually, automatic self-repair techniques are becoming a necessity. Self-repair by automatic replacement puts the spares, associated with any computer installation, to use in effecting extreme system reliability rather than having them sitting idle on a shelf waiting to be manually put into use.

In computing systems of the future, the entire task of fault detecting, identifying, locating, isolating, repairing, and process restoration can be automated by replacement switching with present day state-of-the-art techniques. Future computing systems will undoubtedly incorporate some form of self-repair because many applications require continuous error-free operation.

The many-sided advantages of self-repairing automata for the computing field include:

- Continuous system operation
- 100 percent system availability
- Long term remote system operation (operation for years)
- Reduced maintenance costs.

A computer, through a system design encompassing both hardware and software techniques, can be designed as a self-repairable space and time redundant system. Since the technology now exists to achieve a self-repairable system and such a system is realizable, 100 percent system availability can be virtually assured.

REFERENCES

1. Joseph, E. C., "Multiprocessing for Information Systems", Sperry Engineering Review, vol. 17, no. 1, Summer 1964, pp. 39-43.

2. Burke, T. E., and Wang, G. Y., "Spaceborne Multiprocessing Organizations", WESCON/66, Session 9, "Advanced Spaceborne Computer Concepts", August 23-26, 1966.

3. Cubert, J. S., Simmons, G. T., and others, "Impact of Batch Fabrication on Future Computers", Proceedings of the National Symposium sponsored by the Computer Group of the IEEE, April 6-8, 1965, Los Angeles, California.

4. Champine, G. A., and Griffith, G. M., "Automatic Error Recovery in the Nike-Zeus Guidance Computer", Fall Meeting of ACM, 1962.

5. Pierce, W. H., "Failure-Tolerant Computer Design", Academic Press, New York and London, 1965.

6. "On-Line Computing Systems: A Summary", Proceedings of the Symposium On-Line Computing Systems, Los Angeles, California, February 2-4, 1965.

7. Kneale, S. G., "Reliability of Parallel Systems with Repair and Switching", Proc. Seventh National Symposium on Reliability and Quality Control, IRE, Philadelphia, Pennsylvania, January 9-11, 1961; pp. 129-133.

8. Einhorn, S. J., "Reliability Prediction for Repairable Redundant Systems", Proceedings of the IEEE, February 1963, pp. 312-317.

ACKNOWLEDGEMENTS

In presenting this paper, the author wishes to express his appreciation for the assistance and cooperation offered by fellow colleagues at UNIVAC. Certainly, an effort as complex as that discussed herein is the result of many individual contributions. A special expression of gratitude, however, is owed to Mr. D. R. Lewis for providing many comments which served to refine this work.

In addition, acknowledgement is due to Mr. H. J. Corning for his reliability analysis and simulations which aided and confirmed the results reported. Finally, the author wishes to express his debt of gratitude to the other staff members whose untiring efforts made this paper possible.

ARITHMETIC ERROR CORRECTION

HARVEY L. GARNER

Dr. Garner has been a Professor of Electrical Engineering at the University of Michigan since 1963. He received his B. S. and M. S. (Physics) degrees from the University of Denver in 1948 and 1951, respectively, and his Ph. D (Electrical Engineering) degree from the University of Michigan in 1958.

Dr. Garner was a Research Associate, Cosmic Ray Research Program, University of Denver, and the Inter-University High-Altitude Research Laboratory from 1949 to 1951. He was Chief Engineer, Engineering Research Institute, University of Michigan, where he was involved in the development and operation of the MIDSAC and MIDAC computers from 1951-55.

He was an Instructor in Electrical Engineering at Michigan (1955-58) and Chairman of the intensive summer computer courses (1955 to present). He became an Assistant Professor in 1958, an Associate Professor in 1960, and a full Professor in 1963 at the University of Michigan. In addition, he was Director, Information Systems Laboratory, at the University from 1960 to 1964.

Dr. Garner is a member of the Communication Sciences Program Committee (1960 to present), was an Organizer for the Computer Sessions, 1962 IRE Convention, is a member of the Board of Directors of the Ann Arbor Computer Company, 1965-1966, a Director of the Systems Engineering Laboratory at the University (1965-1966), and Acting Chairman for the Program in Communication Sciences at the University from September 1965 to present. In addition, he has been a consultant to IBM, Lockheed, Bell Telephone Labs, Ford Instrument, Strand Engineering, Holley Carburetor, Sylvania, Cincinnati Milling Company, ESSO Production Research, and the U. S. Air Force.

N 67-17107

ARITHMETIC ERROR CORRECTION

by Harvey L. Garner

Professor of Communication Sciences
and Electrical Engineering
The University of Michigan
Ann Arbor, Michigan

SUMMARY

In this paper the classification and properties of arithmetic codes is briefly reviewed. The theory of error checking and correction is well developed. However the application of this theory to practical computers has been limited because of the effects of error checking on computation rate or because of the relative complexity of error control circuits in comparison with arithmetic circuitry. It is possible that some applications of error codes have lead to a less reliable overall unit because of the complexity of the error control equipment which is also subject to malfunction. It is well known that a modulo three residue check suffices for the detection of all single arithmetic errors. In this paper a new logical circuit for the determination of the modulo 3 residue is presented and the expected performance of this circuit is analyzed, using the assumption that the error events follow a binomial distribution. The modulo three check circuit operates faster than the conventional adder carry logic and the statistical analysis indicates that the circuit is practical.

Introduction

Error correcting codes for arithmetic operations have received considerable attention. Peterson has shown that all separate checking codes are residue codes[11]. Brown has introduced the AN codes [1] and Henderson has given examples of systematic codes [6,7]. Since Henderson does not consider the code arithmetic, it is impossible to determine whether these codes as presented by Henderson were meant to be separate or nonseparate. Recently Garner [3] has shown that both the Brown codes and the Henderson codes are members of the same general class of nonseparate codes which are ideals contained in rings of integers.

Error codes are classified according to three characteristics: (1) Parity or Residue check, (2) Separate or Non-separate arithmetic for the check digits and the number digits, (3) Systematic.

A single malfunction in the conventional adder logic produces errors represented by burst error patterns for parity codes. The burst length or weight may have any value from zero (no error) to n+1 (all digits of the sum in error)[5]. All burst errors due to single component malfunctions in a standard binary adder are obtained if + is used to combine the error patterns of the type $e = +2^{j-1}$, $j=1,2,\ldots,n$, with the correct sums [1]. Thus, error analyses for single component malfunctions can be simplified if + is used to combine the error patterns since only 2n patterns need be considered.

A parity check checks only the digitwise modulo addition in the addition process. Specifically, it does not check the carry generation process. Errors in carry generation will not be detected. Any of the transmission type of error correcting codes can be used, rather than the simple parity check. However, such codes will still only obtain error correction for errors in digitwise modulo addition[13]. The absence of a check on the carry generation process plus the burst nature of the error patterns tend to render the parity check useless. Garner [4,5] compares the effectiveness of a two bit parity check against a modulo three residue check. The modulo three check is shown to detect all errors due to single component malfunction while the two bit parity check detects at most 92% of these errors.

A separate code for a nonredundant number system N is γ, the single-valued mapping defined by the set of ordered pairs n, $\gamma(n)$ such that each $n \in N$ occurs in one and only one ordered pair. This mapping is indicated by $\gamma: N \rightarrow R$. $\gamma(n) \in R$ is the check digit for $n \in N$. The separate code is characterized by the absence of any arithmetic interaction between N and R. Different arithmetic is defined for N and R. A theorem due to Peterson[11] states that every separate check code is a residue code or is isomorphic to a residue code. Separate residue codes, for binary numbers, have the undesirable requirement of sign correction if twos complement coding is used. If ones complement coding is used, then sign correction is not required.

Nonseparate codes and transmission codes have the same basic structure. Let Q be the set of all distinct n tuples over $\{0,1\}$. Then K, the nonseparate code, is a subset of Q. A single arithmetic unit processes the coded elements of K. Thus check arithmetic and operand arithmetic are not separated. The general class of nonseparate codes includes the AN codes but is not restricted to a diminished radix complement interpretation for K. This is desirable since other complement interpretations of K can be easily realized and these avoid complete end around carry correction.

A nonseparate code is systematic if for each $n \in N$ there exists a unique $k \in K$ such that n can be identified in k. Separate codes are trivially systematic. An example of a systematic code was given by Henderson [6, 7]. The code

arithmetic was not discussed. Henderson indicated that the code should be a concatenation of $g - |n|_g$ and n. This description leads to the use of the term "additive" to describe this class of codes. The term additive is not appropriate since the Henderson code is a member of the general class of nonseparate codes. All codes in this class are multiplicatively generated. An important property of the systematic, separate codes is that multiplication requires no correction.

The smallest value of a check base for single-error detection for binary arithmetic is three. The separate code structure is preferred for a one's complement code if n, the number of bits, is even. If n is odd, a separate code does not exist for g=3 but a systematic, nonseparate code exists. For the two's complement code, the nonseparate systematic code structure exists for all code lengths if g=3; and the nonseparate structure is preferred over the separate code structure since sign and multiplication correction are not required [3].

The basic difficulty relative to the application of error detecting or correcting codes of the separate or nonseparate type is the number of the components or the time required to determine the check and effect the correction. The check or correction computation must be accomplished in about the same time required for addition. Some solutions to this problem can be obtained by using properly coded stored tables. The actual effectiveness of check realizations has received little attention. Major research efforts have been devoted to the structure of the codes. A preliminary study of the utility of a modulo three checker is considered in this paper.

Bounds on Arithmetic

We shall consider the following as basic time units relative to the computational period of a logical circuit

τ_f = min period between successive input pulses resolvable by a flip-flop

τ_g = the delay associated with a single unit (i. e. a transistor in a logical gate)

Technology is such that $2\tau_g \leq \tau_f, \leq 4\tau_g$ [9].

In the following discussion, let $k\tau_g = \tau_f$. τ_A, the period of one addition operation for an accumulator type adder, can be defined as the period between the set of the input register and the set of the accumulator. For a conventional ripple carry adder $\tau_A \cong 2n\,\tau_g$. Thus

$$\tau_f < \tau_A \leq 2n\,\tau_g = \frac{2n\,\tau_f}{k} = \tau_{AR} \qquad (1)$$

A faster adder is obtained with an "exclusive OR carry"[3,14]. The upper bound is reduced to at least $\frac{n\tau_f}{k}$. In fact, $\tau_{A\oplus}$ should approach τ_f. Lehman [10] has made a detailed comparative study of the cost and computation time for all known adder configurations. His results show the adder with exclusive OR carry generation to be as much as seven times faster than the conventional ripple carry adder. Specifically

$$\frac{\tau_R}{7} \leq \tau_{A\oplus_2} \leq \frac{\tau_R}{3} \qquad (2)$$

An adder with a modulo four exclusive OR carry generation is as much as fourteen times faster than the ripple carry adder.

$$\frac{\tau_R}{14} \leq \tau_{A\oplus_4} \leq \frac{\tau_R}{6} \qquad (3)$$

These substantial reductions in τ_A are associated with only fractional increases in hardware.

	Components Per Stage		Total Semi-Conductor
	diodes	transistors	devices/stage
Ripple carry	6	2. 2	8. 2
Exclusive OR (Mod 2)	6	6	12
Exclusive OR (Mod 4)	10	4. 6	14. 6

TABLE 1

Components per Stage for Different Carry Schemes (Lehman [10])

The count in Table 1 does not include the two flip-flops per stage in the input and output registers. Each flip-flop consists of eight semiconductor devices. Thus the different adder configurations require approximately 24, 28, 31 semi-conductors per stage.

Thus, modest increases in the number of components (17% to 25%) yield an adder such that

$$\tau_f \leq \tau_{A\oplus} \leq \frac{2n\,\tau_f}{kq} \qquad (4)$$

where $3 \leq q \leq 14$ and $6 \leq kq \leq 64$. The upper bound is valid only if

$$\frac{2n}{kq} > 1. \qquad (5)$$

The preceding discusion shows the existence of adder designs, requiring a reasonable number of components per stage, with an addition period,

$$\tau_A = \alpha\,\tau_f, \qquad (6)$$

where α is small, but $\alpha > 1$. Multiplication and division are obtained by a sequence of add-shift operations in the conventional parallel arithmetic unit. Even if the add is deleted for zero multiplier digits and multiplier recoding is employed, the multiplication period for τ_M for n-bit operands has a lower bound given by $n\tau_f$. This bound can be lowered further only by using multiple stage shift logic in the accumulator register which is costly. It should be possible to approach the lower bound using a carry store adder. A multiplier using multiplier coding, deletion of add for zero multiplier digits, and single stage shifting in conjunction with an adder using an exclusive OR carry circuit will have an average lower bound for the multiplication period equal to

$$\frac{2}{3}n\tau_f + \alpha\frac{n}{3}\tau_f = n\tau_f\left(\frac{2+\alpha}{3}\right) \qquad (7)$$

SRT division can be considered to have approximately the same lower limits as those given for the above multiplication periods.

Properties of a Switched Mod 3 Adder

The simplest circuit for checking is a modulo 3 adder. The input of the modulo 3 adder is obtained from a switch which samples pairs of adjacent accumulator digits in sequence. Such a circuit requires two flip-flops. The total number of semi-conductor devices required for n stages is x + 2n where $30 \leq x \leq 40$. This count includes the commutator part of the switch but assumes the required pulses for required commutation are available.

Let τ_{CS} be the time period required for a check of n stages. Then

$$\tau_{CS} \geq \frac{n}{2}\tau_f \qquad (8)$$

The checker should almost obtain the lower time bound. The performance of this checker relative to the different adders previously discussed is such that

$$\tau_{A\oplus_4} < \tau_{A\oplus_2} < \tau_{A_R} \leq \tau_{CS}. \qquad (9)$$

Assume that

$$\tau_{CS} = \frac{n}{2}\tau_f. \qquad (10)$$

Previously, we have shown

$$\tau_{AR} = \frac{2n}{k}\tau_f, \quad 2 \leq k \leq 4 \qquad (11)$$

so

$$\frac{n}{2}\tau_f \leq \tau_{AR} \leq n\tau_f. \qquad (12)$$

Thus, under the most optimum circumstances, the switched checker will not add to the addition time of a ripple carry adder. For example, a 10 n. s. carry propagation time per stage requires flip-flops in the checker clocked at 50 Mc. It has been estimated that

$$\frac{\tau_{AR}}{14} \leq \tau_{A\oplus} \leq \frac{\tau_{AR}}{3} \qquad (13)$$

Thus the switched checker will require a substantial period of time over and above the addition period of an adder using an exclusive OR carry circuit. If the required check period can be overlapped with some period when the adder is idle, then the switched checker can be used. However, when this cannot be done, a parallel checker is required.

A Parallel Modulo 3 Check Circuit

Several parallel designs are possible. The design presented here utilizes the same principles used to obtain fast carry propagation for the exclusive OR carry type of adder. The proposed parallel modulo three checker should require a check period no longer than the maximum carry propagation period of the modulo 4 exclusive OR carry circuits, and the hardware realization of both circuits will be subject to identical considerations or restrictions. Thus

$$\tau_{CP} \cong \tau_{A\oplus_4} \qquad (14)$$

Approximately $\frac{n}{2}(15)$ transistors are required to realize this parallel modulo three checker.

Basically the checker consists of a chain of $\frac{n}{2}$ switch units and $\frac{n}{2}$ switch control units as shown in Figure 1. Each switch unit is a 3P3T switch as shown in Figure 2. Each 3P3T switch requires nine transistors and each switch controll logic unit requires six transistors. The checker requires a relatively short period for the check because all switches are set simultaneously shortly after the accumulator is set.

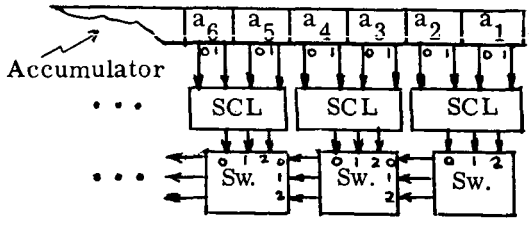


Figure 1

Block Diagram of Parallel Mod 3 Checker

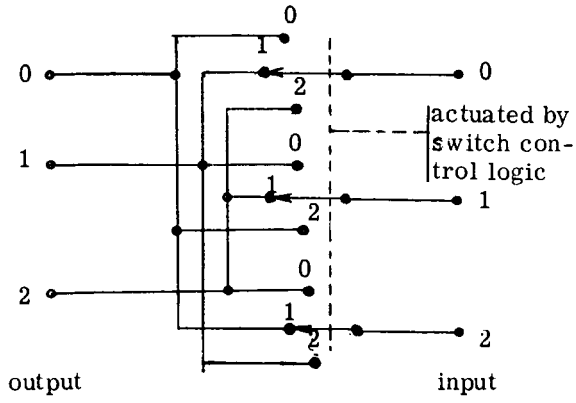


Figure 2

Connections Used in the Parallel Modulo Three 3P3T Switch Checker

The parallel checker should be capable of checking a sequence of additions without requiring excess time for checking. However, on the average, the period for multiplication with multiplier coding and suppressed addition for zero digits is

$$\tau_M = n\,\tau_f\left(\frac{2+\alpha}{3}\right). \tag{15}$$

The checker will require

$$\tau_{CM} \cong n\,\alpha\,\tau_f \tag{16}$$

since a check should occur, after each add-shift operation and after each shift operation in the multiplication process, unless reliability requirements permits checking after the final product. The characteristics of the non-separate, systematic class of residue check codes facilitates the direct checking of products.

Three alternatives are available relative to the check of multiplication using the parallel checker. (1) Check every step and an increase in the multiplication period occurs due to checking unless $\alpha = 1$. In this case there is no need to employ multiplier coding or suppressed addition for zero multiplier digits since

$$\tau_M = \tau_{CM} = n\,\tau_A \tag{17}$$

(2) Check the final product with no check at any intermediate step. No excess time is required. (3) Check each add-shift, shift-sequence during the next add-shift step. A shift operation requires a period approximately equal to $\alpha\,\tau_f$, the same as an addition check. It is expected that this scheme is optimum since the average number of shifts between each add-shift operations is two and the shift operation is more reliable than addition since less time and fewer components are required.

A checker for both the high order and the low order part of the accumulator is required unless only rounded products are required.

Evaluation of the Checked Adder

Assume each semi-conductor has a probability of failure $P = 1-Q$ in the time interval τ_f. Successive time intervals are assumed independent. The probability of at least one failure in an unchecked arithmetic unit containing r semi-conductors in τ_f is

$$p_1(F,\tau_f) = 1 - p(0,\tau_f) = 1 - P^0Q^r \cong rP, \quad p < 1 \tag{18}$$

A checked arithmetic unit requires r + m components. Let $p_2(F,\tau_f)$ denote the probability in the checked arithmetic unit in τ_f and

$$p_2(F,\tau_f) \cong (r+m)P \tag{19}$$

The probability of at least one failure of a check circuit semi-conductor in τ_f is

$$p_3(F,\tau_f) \cong mP \tag{20}$$

Thus an upper bound on the fractional increase in errors due to the checker is given by

$$f_C(e) < \frac{m}{r}. \tag{21}$$

Using the component counts for the various adders and the parallel checker fixes $f_C(e)$ between 1/4 and 1/3. This upper bound is not realistic because the checker for the adder also checks other components in the computer; i. e. memory and data transmission between memory and the arithmetic unit.

The modulo three checker will correct all errors due to a single component malfunction in τ_f and 1/2 the errors due to two component failures in τ_f. So, with checking, the probability of an undetected failure in τ_f is

$$p_u(F,\tau_f) < 1 - Q^t\left(\left(\frac{P}{Q}\right)^t + t\left(\frac{P}{Q}\right) + \frac{t(t-1)(P)^2}{4\,Q}\right)$$

where $t = r + m$.

The ratio of the probability of an undetected failure in τ_f with and without checking is

$$\frac{p_u(F,\tau_f)}{p_1(F,\tau_f)} < \frac{1 - Q^t\left(t\frac{P}{Q}\left(1 + \frac{t-1}{4}\frac{P}{Q}\right)\right)}{1 - Q^{t-m}}$$

Two complete adders with parallel checking and a comparator between the accumulators can be realized with 2t + 2n semi-conductors. All errors are detected unless there is a component malfunction and the error is correctable except when an error not detectable by the parity checker occurs or when a detectable error occurs in both adders. Erroneous correction can occur because of the possibility of an uncheckable error in one adder coupled with a checkable error in the second adder. Evaluation of this configuration is in process.

(This research was partially supported by Air Force Contract AF 30(602)-3546.)

Bibliography

[1] D. T. Brown, "Error detecting and correcting binary codes for arithmetic operations," IRE Trans. on Electronic Computers, vol. EC-9, pp. 333-337, September 1960.

[2] C. V. Freeman, "Statistical analysis of certain binary division techniques," Proc. IRE, vol. 49, no. 1, pp. 91-103, September 1958.

[3] H. L. Garner, "Error codes for arithmetic operations," IEEE Trans. on Electronic Computers, vol. EC-15, pp. 51-57, October 1966.

[4] H. L. Garner, "Generalized parity checking," IRE Trans. on Electronic Computers, vol. EC-7, pp. 207-213, September 1958.

[5] H. L. Garner, "Error checking and the structure of binary addition," Ph. D. dissertation, The University of Michigan, Ann Arbor, Michigan (1958).

[6] D. S. Henderson, "Logical designs for arithmetic units," Ph. D. dissertation, Harvard University, Cambridge, Massachusetts (1960).

[7] D. S. Henderson, "Residue class error checking codes," Proc. 16th National Meeting of the Assoc. for Computing Machinery, Los Angeles, California (1961).

[8] T. Kilburn, et al., "A parallel arithmetic unit using a saturated-transistor fast-carry circuit," Proc. IRE, vol. 107, B. 36 pp. 573-584 (1960).

[9] M. Lehman, et al., "Serial arithmetic techniques," AFIPS Conf. Proc. of the Fall Joint Computer Conference, vol. 27, Part 1, pp. 715-725 (1965).

[10] M. Lehman, "A comparative study of propagation speed-up circuits in binary arithmetic units," IFIP Congress Proc., Munich, Germany, pp. 302-304 (1962).

[11] W. W. Peterson, "On checking an adder," IBM J. Res. and Dev., vol. 2, pp. 166-168 April 1958.

[12] J. E. Robertson, "A new class of digital division methods," IRE Trans. on Electronic Computers, vol. EC-7, pp. 218-222 September 1958.

[13] J. E. Robertson, "Error detection and correction in binary parallel digital computers," in Electronic Digital Computer, Internal Rept. 37, Digital Computer Lab., University of Illinois, Urbana, pp. 70-81 (1952).

[14] F. Salter, "High-speed transistorised adder for a digital computer," Trans. IRE, vol. EC-9, 4, pp. 461-464 (1960).

SYSTEM ORGANIZATION OF THE JPL SELF-TESTING AND -REPAIRING COMPUTER AND ITS EXTENSION TO A MULTIPROCESSOR CONFIGURATION

ALGIRDAS AVIŽIENIS

Dr. Avižienis received B. S., M. S., and Ph. D degrees in Electrical Engineering from the University of Illinois, Urbana, in 1954, 1955, and 1960, respectively. In 1955-56 he was a research engineer at the Jet Propulsion Laboratory, Pasadena, California. During graduate studies at the Digital Computer Laboratory, University of Illinois, he was a Fellow in 1954-55, 1956-57 and 1957-58, and a research assistant in 1959-60, participating in the design of the ILLIAC II system. He also was a staff engineer at Barnes and Reinecke, Inc., Chicago, Illinois, in 1958-59.

In 1960 he rejoined the Jet Propulsion Laboratory as a Senior Engineer in computer systems research and initiated the JPL Self-Testing and Repairing (JPL-STAR) Guidance Computer research project. Since 1962 he has been an Assistant Professor of Engineering at the University of California, Los Angeles, conducting research in computer arithmetic and digital system design in connection with the Variable Structure Computer project. He has also remained associated with the Jet Propulsion Laboratory as Principal Investigator of the JPL-STAR Computer Research project. In the spring of 1966 he was a visiting professor at the National Computing Center of the National Polytechnic Institute of Mexico, Mexico City, where he assisted in the organization of a graduate program in computer sciences.

Dr. Avižienis is a member of Sigma Xi, Tau Beta Pi, Eta Kappa Nu, the ACM, and of the Technical Committee on Switching and Automata Theory of the IEEE Computer Group.

SYSTEM ORGANIZATION OF THE JPL SELF-TESTING AND -REPAIRING COMPUTER AND ITS EXTENSION TO A MULTIPROCESSOR CONFIGURATION

By Algirdas Avižienis

NASA Jet Propulsion Laboratory, Pasadena, California
and
University of California, Los Angeles, California

SUMMARY

The techniques for the application of protective redundancy in digital systems are reviewed and compared. The choice of a replacement system as protective redundancy for a spacecraft guidance and control computer is reached by consideration of its computing requirements. The system organization of an experimental replacement system is described, with emphasis on the method of concurrent fault diagnosis by means of arithmetical encoding. An extension of the system to a multiprocessor configuration is considered as a means to provide on-board data processing after arrival at a remote destination.

INTRODUCTION: RELIABILITY BY MEANS OF PROTECTIVE REDUNDANCY

Reliable performance of digital systems is usually attained by the systematic application of two techniques. The first is the selection of highly reliable components and the use of proven methods for their interconnection and packaging. The second technique is an extensive verification of the design and of the programs, first by simulation and later by diagnostic and functional tests under expected environmental conditions. Despite of these reliability assurance techniques, the system may still fail during use because of uncontrollable or undetected faults. These include undetected design errors, random failures of components or connections, and externally induced failures due to the environment (nuclear radiation, sparks, mechanical damage, etc.).

The effects of these faults can be controlled by the third reliability technique - the introduction of protective redundancy into the system. A computer system contains protective redundancy if the effects of component failures or program errors can be tolerated because of the use of additional components or programs, or the use of more time for the computational tasks. These additional components, programs, and time are not required by the system in order to execute the specified tasks as long as no failures or transient malfunctions occur. The techniques of protective redundancy may be divided into two major categories: massive (also called masking) redundancy and selective (also called stand-by) redundancy.

In the massive redundancy approach the effect of a faulty component, circuit, signal, subsystem, program, or system is masked instantaneously by permanently connected and concurrently operating replicas of the faulty element. The level at which replication occurs ranges from individual circuit components to entire systems. The principal techniques of massive redundancy are:

1. Replication of circuit components; e.g., "quadded" diodes, resistors, transistors; duplicated connections, etc. (Refs. 1, 2).

2. Replication of logic signals: use of multiple channels and voting elements, recursive nets, interwoven logic, variation-tolerant threshold element nets. (Refs. 3, 4, 5, 6, 7).

3. Adaptive logic elements, e.g., voters with variable-weight inputs. (Ref. 6)

4. Replication of entire systems with comparison and voting or diagnosis at system level.

In the selective redundancy approach the presence of a faulty element is detected by observing a symptom of the failure; subsequently the fault is made harmless by a corrective action. The principal techniques of selective redundancy are:

1. Error detection and correction using error-correcting circuits for coded words. (Refs. 8, 9)

2. Replacement of the faulty element or system by a stand-by spare (self-repair).

3. Reorganization of the system into a different computer configuration. (Multiprocessors and other "degradable" systems)

The last two methods presuppose the existence of a diagnosis procedure which will recognize the symptoms of a fault (Refs. 10, 11), and of a switch which implements the replacement or reconfiguration. (Refs. 12, 13). Error correction is attained by recomputation, possibly retracing several steps in the program to a "rollback" point.

APPLICATION OF PROTECTIVE REDUNDANCY IN A SPACECRAFT GUIDANCE COMPUTER

The choice of a method or of a combination of methods from the preceding list for a particular computing system is influenced by the intended application. The present paper considers the application of protective redundancy to a guidance and control computer for an unmanned

spacecraft which may also be employed for the onboard processing of scientific data when guidance computation is not in progress. The guidance computer is required to survive space voyages to other planets which range up to several years in length and to perform approach guidance and control computation at the end of the voyage. Continued control of the spacecraft after arrival may also be required. Course corrections are to be computed one or more times during the voyage; considerable time is available for this task. The computing at launch and in early stages of the voyage may be performed or supported by computers on the ground and in the launch vehicle. The extreme distance and the potential occultation make ground support less effective at approach to the planet, therefore the approach presents the most severe problems to the guidance and control computer.

The computer design must also be performed within the constraints of the available power, weight, and volume. The existence of these constraints indicates an advantage for selective redundancy, which does not necessarily require power for the spare replicas and which offers protection with the minimum of one spare for each operating element. On the other hand, the principal advantages offered by the massive approach are:

1. The corrective action is immediate and "wired-in"; it is delayed and requires switching in selective redundancy.

2. During operation there is no need for diagnosis, which is essential in selective redundancy.

3. All parts of the system are equally protected; unprotected "hard core" elements may exist only at interfaces with other systems. In selective redundancy schemes a "hard core" always exists in the system.

4. The conversion of a non-redundant design to a massively redundant one is relatively straightforward; more novel design techniques are demanded by the introduction of selective redundancy.

Compared to massive redundancy, the selective form requires several additional features: a system ability to tolerate interruptions for repair and to execute a "rollback" for error correction, sophisticated diagnosis methods, protection for the "hard core", and trade-off studies between time, program, and hardware replication. The advantages of selective redundancy over the massive form are, however, also very significant in our application:

1. Power is required by only one copy of each replaceable item in a replacement system; all copies require power in the massive form.

2. The replacement switch provides fault isolation between subsystems; such isolation is essential in the case of catastrophic failures. Massive redundancy usually assumes independent failures of logic elements; such independence requires isolation which is difficult to provide for integrated circuit packages which are batch-fabricated and contain many logic circuits in close proximity. The entire batch may possess the same defect; also, mechanical or thermal damage is likely to affect an entire package, rather than single logic circuits.

3. All spares can be utilized in selective redundancy; in the massive form a majority of faulty elements in a given region leads to system failure.

4. The designs of individual replaceable blocks may be altered, and the number of spares may be adjusted to a given mission without changes in the system design in the case of selective redundancy; such changes are more difficult in the massive case.

5. The replication in massive redundancy frequently leads to increased fan-out and fan-in requirements for logic elements, or to increased tolerance limits in circuit design; such problems are avoided in the selective case.

6. Permanent connection of the redundant elements makes the pre-mission check-out more difficult to implement in systems with massive redundancy; special circuits and system outputs are necessary.

7. Massively redundant systems with voting require synchronization of the separate channels at the voting elements; they also are susceptible to transient external influences (e.g., sparks) which alter logic signals in a majority of channels without leaving permanent damage. The delayed occurrence of diagnosis in the selective case allows detection of such transient changes in signals.

CHOICE OF REDUNDANCY TECHNIQUES FOR THE JPL-STAR COMPUTER

Evaluation of the differences between the massive and selective approaches led to the choice of selective redundancy for the protection of an experimental prototype for a spacecraft guidance computer, which will be called the "JPL Self-Testing and -Repairing" (abbreviated JPL-STAR) computer in this paper. The requirement for approach guidance demands a certain computing capacity at the end of a long voyage, and there is no anticipated requirement for a higher capacity at an earlier time. Under these conditions, a replacement system possessing the required capacity is preferred over a reorganizable or "degradable" system which has a minimal

configuration of the same capacity. The replacement system avoids the programs, switches, and control hardware which perform the reconfiguration and resulting rescheduling of programs.

The diagnosis, or self-test, is an essential function of a replacement system. The most common approach - periodic diagnosis - utilizes a diagnostic program which is stored in the memory. Computation is periodically interrupted and the diagnostic program is executed. Detection of a fault initiates the replacement procedure; the program is "rolled back" to a point preceding the previous (successful) diagnosis period. Errors which have been induced by transient fault conditions remain undetected. The cost of diagnosis consists of the storage used for the diagnostic program, of the time consumed by its execution, and of the time needed for repair and repeated execution of the program segment which was run after the last diagnosis. Such time costs are very severe in approach and re-entry guidance and control programs, which require real-time computing.

The alternate diagnosis method is concurrent diagnosis in which error-detecting codes are employed to show the presence of faults. The execution of every instruction is checked immediately; instead of a stored diagnostic program, the cost includes the logic circuits which perform the code check. Errors due to transient faults are detectable, and the immediate detection of a fault permits a very short rollback in the program. For these reasons concurrent diagnosis is preferable in the JPL-STAR computer.

The simplest and most costly code (100% redundancy) is the complete duplication of program and data words. Errors are indicated by the disagreement of two words; diagnosis is needed to pinpoint the faulty source. Parity and many classes of more complex codes which detect errors in the transmission of digital data have much lower redundancy, but are not suitable for the checking of arithmetic operations. In order to have a uniform code for the entire system, arithmetical error detecting codes were selected as a means of diagnosis for the JPL-STAR system. An extensive theoretical investigation of the effectiveness, cost, and applicability of arithmetic codes was conducted prior to the system design of the JPL-STAR computer. (Refs. 14, 15). The results showed the existence of a class of low-cost codes with sufficient effectiveness of error detection.

SYSTEM DESIGN OF THE JPL-STAR COMPUTER

The JPL-STAR computer is a replacement system, which is intended to serve as a prototype for spacecraft guidance computers in very long missions of several years duration. The system consists of several autonomous functional units, including:

1. an arithmetic processor;
2. an index arithmetic processor;
3. a read-only memory;
4. a read-write memory;
5. input/output buffer registers.

The functional units are interconnected and controlled by the central control unit (CCU). One or more replicas of each operating functional unit are included in the system as standby replacements. Replacement of a functional unit is initiated by the CCU and implemented by a replacement switch, which selects the spares in cyclic order. To facilitate checkout, the switch returns to the original operating unit when the spares are exhausted. In order to reduce the size of the switch, all words (instructions and numeric data) are transmitted between the functional units in bytes of four binary digits each.

Word Formats

The arithmetic coding which is most effective in the case of transmission and computing by four-bit bytes employs the check constant 15. (Ref. 15). Any single determinate fault (logic value "stuck on zero", or "stuck on one") will be detected for word lengths up to 14 bytes (56 bits), even if every byte is separately affected by the fault. Binary numerical operands X (28 bits long) are encoded in the product code Z = 15X, yielding 32 bits long code words. The checking algorithm computes the modulo 15 residues (designated as $/Z/_{15}$) of operands and results which are transmitted between the functional units. Error detection is implemented by the checker: a four-bit adder with an end-around carry which sums the bytes of the word being transmitted to obtain the modulo 15 residue. The checkers are located in the CCU; their operation is verified by complete duplication. A non-zero residue is the symptom of a fault in the unit which delivered the operand.

Instructions of the JPL-STAR computer consist of two four-bit operation codes and a 24-bit address part. The address part is also subject to arithmetic operations (addition and subtraction) during indexing and during incrementing of the address. In the selection of a memory location the address is usually divided into two or three segments, which serve as inputs to selection tree networks. Product coded numbers cannot be separated in this fashion into properly coded segments; therefore residue coding with the check constant 15 is used for the address part. In the residue coding, the 20-bit binary address A carries along a 4-bit check symbol c(A), which is the 15's complement of the modulo 15 residue of the address A:

$$c(A) = 15 - /A/_{15}$$

Passing both A and c(A) through the checker should yield the check result 1111, which

represents the zero residue of a product-coded operand. It is very important to note that the residue $/A/_{15}$ as the check symbol will not give the same error-detecting effectiveness as the product code 15X in the case of four-bit bytes, while $15-/A/_{15}$ offers the same effectiveness as the coding 15X.

Two operation codes of four bits each are used in order to have maximal autonomy of the function units. The first code is the control code which remains in the CCU and serves to define the path for the second code - the function code, which is delivered to a function unit designated by the control code. The operation codes are protected by a two-out-of-four encoding, which leaves six valid words in a four-bit code. Such coding is most efficient for short words and is acceptable because operation codes are not subjected to arithmetic operations. It is evident that their validity test is made by a separate circuit, since it cannot be verified by the checker (which is bypassed by the op. codes).

Arithmetic Processors

The main arithmetic processor (MAP) of the JPL-STAR system accepts six function codes: Clear Add, Add, Subtract, Multiply, Divide, and No Operation. (Ref. 16). The operands and results are 32 bit product-coded binary numbers. All arithmetic control is contained in the MAP; an input consists of a function code followed by a coded operand, and the output is a coded result followed by a non-numerical 2-out-of-4 code byte, indicating either one of three singularities (sum overflow, quotient overflow, zero divisor) or the type of a good result (positive, zero, negative). The good result codes are stored in the central control unit (CCU) and are used as data for conditional jump instructions. All partial and final results are delivered to the CCU checker and also stored in a Duplicate Accumulator register in the scratchpad (read-write) memory. A Store instruction is therefore not needed for the MAP. There are four data input lines, four data output lines, and four control lines between the MAP and the CCU. The control lines are a clock input and three outputs: "perform check", "end of algorithm" and "internal fault". The "end of algorithm" serves as a work request; the "internal fault" is obtained from internal monitor circuits which detect catastrophic failures and internal control faults. A breadboard model of the MAP has been constructed and is undergoing functional tests. Residue coding is also applicable to arithmetical operands. An alternate MAP design for residue coded operands is being prepared for a comparison to the present design.

The index arithmetic processor (IAP) contains the Index Register (IR), the Sequence Register (SR) and an adder. When the 20-bit index word B from the IR is added to an address A, its 4-bit check symbol c(B) is added modulo 15 to c(A). The indexed address and the new check symbol go through a checker to the input lines of the appropriate memory unit. The incrementing (by one) of the current address in SR is performed in exactly the same manner, with 1 being added to A and c(1) = 14 being added modulo 15 to c(A) The incremented address returns to SR through a checker. The input and output lines of the IAP are similar to those of the MAP.

Storage

The read-only memory (ROM) contains the programs and the associated constants for the given mission. The address part allows direct addressing of 2^{20}_{14} locations; the experimental model provides 2^{14} words of 32 bits each, using an assembly of magnetic cores and wires for the permanent storage of binary information. The ROM also contains all necessary peripheral electronics: the address register, access circuits, drivers, sequence control, and the output register. Proper operation is verified by the monitoring of driver currents and by the independent readout of a four-bit check symbol of the address designating the location accessed, which is compared to the check symbol in the address register. This comparison verifies that the storage cell which was specified has actually been accessed. All output words from the ROM are delivered byte-by-byte through the appropriate checker. The present model of JPL-STAR computer includes complete replicas of the ROM as replacements; the replacement of peripheral electronics without discarding the core and wire assembly is now being explored. Integrated-circuit ROM's which are presently being developed by several manufacturers promise a considerable reduction in the size and weight of the ROM. The cost of replacement of entire ROM's will be decreased by such miniaturization.

The read-write memory (RWM) and the buffer registers are protected by complete duplication. The RWM provides storage for various intermediate results and inputs; it consists of replaceable core memory modules which contain all peripheral electronics. In case of a permanent fault in one member of a pair, the contents of the good module are copied into a replacement and the faulty module is disconnected. The size of modules and their number is controlled by the total RWM requirements of a mission; the prototype will use a single pair of 128 word modules. It is expected that large-scale integration will provide replaceable RWM modules on one or a few chips. The internal fault monitoring of the RWM modules is similar to the method used in the ROM. It is to be noted that because of complete duplication the copying and replacement may be postponed until a critical computation is completed.

Central Control

The central control unit (CCU) contains the hard core of the replacement system. It serves as the bus connecting all functional units and performs the functions of synchronization (clocking), transferring information, executing the checking algorithm, and implementing

replacement. The transfer of coded words between the functional units occurs on a one-byte (four-bit) bus; it is controlled by the control code of the current instruction. All bytes entering the CCU are directed to a checker. The two-out-of-four code bytes are checked individually by a check network. The bytes of an operand, a result, or an address are summed modulo 15 in the checker and the residue is tested for zero value (represented by four ones) when the transmission is completed. The CCU receives "internal fault" signals from monitoring circuits inside the replaceable functional units, as well as from its own checkers. Two checkers are employed in the present JPL-STAR system configuration: one for outputs of memory units, and one for outputs of the processors.

In the case of a fault signal, the CCU interrupts the current program and executes an emergency sequence. First, the current instruction is repeated in order to correct a transient error; if the fault persists, the replacement switch is advanced. After replacement the program is "rolled back", i.e., resumed at a designated instruction. The address of this instruction is stored in a special CCU register; its updating is a function of the program.

The CCU itself is vulnerable to faults and requires protection. Massive logic or component redundancy (voting, quadding, etc.), complete operating duplication (Ref. 17), periodic self-diagnosis (Ref. 18), and external monitoring are all applicable to this task. Studies are presently being conducted to determine optimal or near-optimal balances of these methods in the CCU. Operational duplication of the checkers and similar functional parts of the CCU permits their replacement and is presently considered as the preferred method of reducing the extent of the hard core.

A highly reliable replacement switch which also provides adequate isolation in the case of catastrophic failures is an essential part of the CCU. A design study which considers magnetic and semiconductor implementations of the switch is in progress. Performance of the switch will be extensively tested under expected environmental conditions.

AN EXTENSION TO MULTIPROCESSING

It was observed in the preceding discussion that the most severe tasks for the spacecraft guidance and control computer occur during approach and re-entry to a planet after a long period of comparative idleness. As a consequence, there is no apparent need to utilize the spares for an extension of computing power by multiprocessing during the earlier phases of the mission, although the spares are available. In general, it is expected that a thorough application of conventional reliability practices will yield a design which is already highly reliable (the longevity of the Mariner IV spacecraft serves as an illustration of this point). The purpose of the replacement system is to provide insurance against overlooked design weaknesses, human errors in production and checkout, and externally induced faults; all of these failures may be catastrophic with respect to an entire functional unit of the system and require the isolation provided by replacement.

Under normal conditions such faults will be avoided, and the replacement system will reach the destination with all or most of its spares still intact. After the execution of the approach and re-entry, the functions of the guidance computer are largely completed; however, there remains a large computational task of processing the scientific data which are acquired during and after the arrival. At this point all surviving spares of the replacement system can be utilized in the new task of on-board data processing, and a multiprocessor configuration becomes desirable.

The conversion of the ordinary replacement system to a multiprocessor requires several additional features. A considerably more complicated bus and switching arrangement is needed to accommodate parallel operation and reconfiguration in case of fault detection. The number of checkers is increased for parallel diagnosis. A more elaborate control unit is needed for the scheduling and coordination of the parallel events. A set of new programs (a new ROM) is also to be provided by the conversion. Design studies of the conversion problem have been initiated with the objective of holding the additional system elements inactive and isolated until the conversion is commanded by the guidance program. Such isolation minimizes the possibility of early system failure in the switch and CCU caused by the additional multiprocessing hardware. After a complete functional checkout the conversion features will be incorporated into the JPL-STAR computer experimental model, which is presently being constructed.

ACKNOWLEDGMENT

The research described in this paper has been carried out at the Jet Propulsion Laboratory, Pasadena, California, under Contract NAS7-100, sponsored by the National Aeronautics and Space Administration. The author wishes to acknowledge the full support and encouragement of W. F. Scott and discussions with J. J. Wedel and G. R. Hansen. The logic design of the main arithmetic processor was performed by A. D. Weeks and D. A. Rennels, and the construction was carried out by J. Buchok, all of the Flight Computers and Sequencers Section, Guidance and Control Division, JPL.

REFERENCES

1. Creveling, C. J.: Increasing the Reliability of Electronic Equipment by the Use of Redundant Circuits. Proceedings of the IRE, vol. 44, pp. 509-515, April 1956.

2. Lewis, T. B.: Primary Processor and Data Storage Equipment for the Orbiting Astronomical Observatory. IEEE Transactions on Electronic Computers, vol. EC-12, No. 5, pp. 677-686, December 1963.

3. Dickinson, M. M., Jackson, J. B., and Randa, G. C.: Saturn V Launch Vehicle Digital Computer and Data Adapter. AFIPS Conference Proceedings, vol. 26, (1964 FJCC), pp. 501-516.

4. Tryon, J. G.: Quadded Logic. Redundancy Techniques for Computing Systems, pp. 205-228, Spartan Press, Inc., Washington, D.C., 1962.

5. Pierce, W. H.: Interwoven Redundant Logic. Journal of the Franklin Institute, vol. 277, No. 1, pp. 55-85, January 1964.

6. Pierce, W. H.: Failure-Tolerant Computer Design. Academic Press, Inc., New York, 1965.

7. Winograd, S., and Cowan, J. D.: Reliable Computation in the Presence of Noise. The M.I.T. Press, Cambridge, Mass., 1963.

8. Peterson, W. W.: Error Correcting Codes. The M.I.T. Press and John Wiley & Sons, Inc., New York, 1961.

9. Kautz, W. H.: Codes and Coding Circuitry for Automatic Error Correction Within Digital Systems. Redundancy Techniques for Computing Systems, pp. 152-195, Spartan Press, Inc., Washington, D.C., 1962.

10. Seshu, S., and Freeman, D. N.: The Diagnosis of Asynchronous Sequential Switching Systems. IRE Transactions on Electronic Computers, vol. EC-11, no. 4, pp. 459-465; August, 1962.

11. Roth, J. P.: Diagnosis of Automata Failures: A Calculus and a Method. IBM Journal of Research and Development, vol. 10, No. 4, pp. 278-291, July 1966.

12. Griesmer, J. E., Miller, R. E., and Roth, J. P.: The Design of Digital Circuits to Eliminate Catastrophic Failures. Redundancy Techniques for Computing Systems, pp. 328-348, Spartan Press, Inc., Washington, D.C., 1962.

13. Avizienis, A.: Coding of Information for a Guidance Computer with Active Redundancy. JPL Space Programs Summary No. 37-22, pp. 9-12, 1963.

14. Avizienis, A.: A Set of Algorithms for a Diagnosable Arithmetic Unit. JPL Technical Report No. 32-546, March 1, 1964.

15. Avizienis, A.: A Study of the Effectiveness of Fault-Detecting Codes for Binary Arithmetic. JPL Technical Report No. 32-711, September 1, 1965.

16. Avizienis, A.: The Diagnosable Arithmetic Processor. JPL Space Programs Summary No. 37-37, vol. IV, pp. 76-80, 1966.

17. Downing, R. W., Nowak, J. S., and Tuomenoksa, L. S.: No. 1 ESS Maintenance Plan. The Bell System Technical Journal, vol. 43, No. 5, part 1; pp. 1961-2019; September 1964.

18. Forbes, R. E., Rutherford, D. H., Stieglitz, C. B., and Tung, L. H.: A Self-Diagnosable Computer. AFIPS Conference Proceedings, vol. 27, part 1, (1965 Fall JCC), pp. 1073-1086.

COMPUTER AIDS

COMPUTER DESIGN ASSISTANCE FOR THE EVOLVING LARGE SCALE INTEGRATED CIRCUIT TECHNOLOGY

JOHN S. MERRITT

Mr. Merritt is a Development Engineer, Computer Systems, Honeywell Corporation. He received a B. S. degree in Electronic Engineering from Rutgers University in 1958, and studied Programming and Theory of Automatic Computation at U. C. L. A.

At Honeywell, Mr. Merritt was responsible for the development of computer design aids using existing computers. Since December 1965, he has been engaged in advanced computer development for an aerospace multi-processor computer system.

Mr. Merritt joined Aero-Florida in 1962 as an electrical engineer, assigned to airborne computer programming for the SAINT project. He was responsible for work on inertial navigation, platform calibration, and assembly integration testing. Later he was involved in simulation work on the H-800 ground computer. Since September 1963 he has programmed definition compiler studies and airborne computer design.

Mr. Merritt was employed as an electrical engineer for Remington Rand (1958-1961), working on such projects as Titan and Nike-Zeus, and was involved in the application of logical design techniques to the design of the logical circuitry between the paper tape reader and the magnetic drum.

Mr. Merritt is a member of Pi Mu Epsilon (Mathematics Honor Society) and Delta Phi Alpha (German Honor Society). His professional writings include: "The Analog Computer," Rutgers Engineer, March 1957.

N 67-17109

COMPUTER DESIGN ASSISTANCE FOR THE EVOLVING LARGE SCALE INTEGRATED CIRCUIT TECHNOLOGY

By John S. Merritt

Electrical Engineer
Advanced Computer Development
Honeywell Inc., Aeronautical Division
St. Petersburg, Florida

SUMMARY

The design of a multiprocessor computer system will be developed with the aid of an existing computer. A tool box of programs is presented which have in common the fact that they all work on the same reel of magnetic tape. This tape contains files each of which specifies the design of a particular unit. The operational steps in execution of selected program tools for a particular equipment design fall into four general categories: 1) Formulation of logic, 2) Simulation of operation, 3) Placement of components, 4) Preparation of wire-run lists.

Each operational step may be thought of as a different shelf in a tool box. On each shelf are several program tools which may be selected by the designer. Tools presently available and planned are described. The objective of these tools is to provide a cooperative man-machine interactive design system which frees the designer from routine bookkeeping tasks so that he may devote more of his time to the actual system design. The elimination of breadboarding and manual layout techniques not only reduces to a few hours computer time the many man-years this takes by hand but with the advent of Large Scale Integrated Circuit Technology provides the most practical method of implementation.

INTRODUCTION

The design of a multiprocessor computer system will be developed with the aid of an existing computer. A tool box of programs has in common the design data to be processed. The evolving design is kept on a reel of magnetic tape. This tape contains files each of which specifies the design of a particular unit. The operational steps in execution of selected program tools for a particular equipment design fall into four general categories:

Phase I - Formulation of Logic

Phase II - Simulation of Operation

Phase III - Placement of Components

Phase IV - Preparation of Wire-Run Lists

Each phase may be thought of as a different shelf in a tool box. On each shelf are several tools which may be selected by the designer. Different tools are independent of one another thus allowing them to be modified or added to without affecting the others. Different types of tools on a shelf are used by the designer to shape the work in the design files. The sequence and use of these tools is determined by the designer according to how the work is developing in the file. Different versions of the same type of tool may be available for processing different integrated circuit building blocks. After each tool has obtained from the file that portion of the design data it works on, its function is performed, and the updated work is returned to the file. Use of tools on subsequent shelves depends upon data processed by tools on the first shelve(s).

However, the same tool can be reused to rework the data after the tool has once been used and data in the file has once reached its level of update. Several tools are never seen by the designer but are used by the tool designer (programmer) in bootstrap development of the program tools and to service and run them. At the same time, the designer uses the tools he requires in bootstrap development of the design data.

Designs which use the same technology use the same program set but new technology designs require program modification. This is done either by making a copy (new version) of an existing program with modifications or by adding a new special purpose program. Thus, the system is expanded without affecting what has already been accomplished.

The program tools now being used for integrated circuit technology are being converted to machine independent programs so that they may run on any computer having the proper capability. The tool box is being expanded to include additional tools for use in processing various large scale integrated circuit approaches.

Following is a description of each computer program of each phase. All programs are run under control of the engineer design group using them. Computer runs are supervised by the design executive program (see "Executive" under Service Programs). One computer run may execute any useful combination of programs. More than one unit design may also be processed on one run.

PHASE I - FORMULATION OF LOGIC

A design in the form of equations specifying both logic and interconnections must be loaded into a file on the magnetic tape. Modifications of the file are made until the logic design has been formulated. Load lists are prepared. Various sections of the design are merged substituting signal symbolic names where necessary to maintain consistency.

The logic design is manually partitioned into tentative sets of logic for LSI building blocks. Composite multi-function truth tables are generated. An automatic commonality analysis is performed with the objective of minimizing the total number of minterms for all functions in the set. The resulting new equation set is then reduced one equation at a time by Boolean simplification techniques. Full use is made of don't care conditions in both the commonality analysis and subsequent Boolean simplification. Checks are made wherever possible in all program applications. Errors are listed but do not stop the design process. Defective data is returned to the file in its original form and is not updated.

Load Logic Equations

This is a basic building block of the computer design assistance system. It loads equations into a new file via punched cards or corrects an existing file on a previous file magnetic tape. A control card specifies input and output options and the name of the design file. Corrections of changes, additions or deletions automatically modify all affected data in the file.

Single valued functions are represented by single equations where the subject symbol represents the output signal and the input symbols represent the input signals. Logic operator connectives and function name indicate the logic function to be performed between inputs and output. Multi-valued functions are represented by sets of single equations. All the subject symbols of a set are identified by a set-name much as a programmer names a subroutine. Characteristics of the set, such as load lists, are identified. Equations which only feed other equations within the set will not appear as outputs from the set and such equations' load lists will be internal to the set. Only those load lists external to the set will be identified as set loads. Set loads are separated by the subjects of set output signals.

A load list is generated for a particular equation by obtaining the subjects of all other equations which have the particular equation's subject as an input. This process is left until the entire file has been updated with additions, changes and deletions. Changes replace an equation's inputs but not its subject and deletions not only completely remove an equation from the file, but also remove the equation's subject symbol wherever it may appear as an input in other equations. After updating the file, load lists are regenerated for the entire file. Those inputs which are not represented in the file by equations cause extensions to be generated which are equations with subject and loads, but no inputs.

Output options include complete listings of equations including loads or any portion thereof, load lists, boundary items (such as extensions) listing signals entering and leaving the design, listings of equations without loads, punched cards of equations suitable for reloading into a file of a new unit which may be a redesigned version of the existing file.

Generate Truth Tables

After the logic designer has loaded equations, he identifies multiple equation sets which he determines may form a useful LSI building block function. After this manual partitioning, composite multi-function truth tables are generated for each set. This is done by a logic simulation program which simulates each set for all combinations of inputs which the logic designer indicates. Input and output binary word pairs are thus generated for each set regardless of the logic contained within the set (see Simulate Logic Equations). All serial logic (that is, equations feeding other equations) within the set will have been transformed into parallel logic because only input and output values will be given in the composite truth table.

Practical limitations will be held to due to LSI pin-out restrictions. The truth tables will go into file and not be printed out unless requested. Manual inputting of truth tables is also possible, and in this case equations need not be in the file for such functions. A future design aid would be a problem oriented compiler to generate truth tables by programming instead of simulation.

Commonality Analysis

Functions within the set which for the same combination of input values obtain the same output value have minterms in common. A minterm is defined as a Boolean product of all input variables, with each variable present in either its true or complemented form depending upon whether or not the corresponding input value is a 1 or 0. Since a function may be represented as a Boolean sum of

all minterms for which the function is true, (that is, 1), only such minterms will be considered together with don't cares.

The objective of commonality analysis is the minimizing of the total number of minterms for all functions in the set. A trial and error approach can be followed to generate a new set of logic equations in expanded minterm form. New functions are generated of each minterm for which all N outputs are true, for which all N-1 outputs are true, all N-2, etc. The resulting new equation set represents the original function when common logic outputs are ORed together with the logic unique to each function.

Boolean Simplification

After the multi-valued function minterm reduction is performed by commonality analysis, the resulting new equation set is then reduced one equation at a time by Boolean simplification techniques. Each equation is represented by its truth table which is a listing of all minterms for which the function's output is true. This minterm form is also called the first canonical form or the disjunctive normal form.

Each minterm is represented by a binary word of N bits where N is the number of equation inputs. The value of the word is determined by those combinations of bits for which the corresponding combinations of input signals obtain a true output. The words of this truth table are then grouped according to the number of 1's in each word. Words of adjacent groups are compared for a match in all but one bit position. Such matches produce a new word of N-1 variables with an X marking the deleted variable. This is equivalent to applying the theorm $xy + \bar{x}y = y$. After N passes, prime implicants will remain. The simplest sum-of-products representation is obtained by the Quine method of Boolean simplification.

This program has been written in Cobol and run for 12 variables in reasonable time on a 131K character H-2200 computer. Any program which takes more than an hour to run is considered unreasonable in execution time.

Substitute Equation Symbols

Input consists of match symbols denoting the equation symbols to be modified and substitute symbols specifying the modification. This program is useful as a clerical aid in changing signal names. If two design files are to be merged, common signals must have the same names and one file may have to substitute symbols. All occurrences of the same symbol in the file are automatically modified. If a character in the match symbol is a hyphen, that character position is omitted in the comparison and substitution. Deletion of symbols is another option.

PHASE II - SIMULATION OF OPERATION

Phase II programs depend upon data processed by Phase I programs. After a design has been formulated, it is tested and checked by Phase II programs. Any errors found can be corrected by rerunning Phase I programs. Circuits are associated with equations by the circuit assignment program in order to allow detailed circuit timing checks to be performed.

Functional Complexity Check

After the reductions of Phase I, multi-valued function sets must be checked to see if the commonality and simplification was enough to allow the function to meet various LSI pin-out limitations and logic functional complexity limitations. Output of this program are error listings. The file is not modified in any way.

Simulate Logic Functions

Besides the logic equations in a design file, the program simulate logic equations also accepts as input a card deck which controls the simulation timing, inputting of test data, output format, and output of selected circuits at selected times. Output is to the high speed printer. The design file is not modified in any way.

The program will handle up to 24,000 logic equations as presently written in assembly language on the H-800 Computer. Run time depends upon number, size and type of equations; average number of unclocked circuits in a chain; number of clock phases in a clock cycle; amount of test data input and amount of output. Although for synchronous logic, the program will accept asynchronous chains logically separated from other chains. Unclocked logic feedback is accepted and logic loops and oscillatory conditions are identified if they exceed maximum allowable iterations. A subroutine exists for each allowable logic equation type. This subroutine library can be updated for new circuit types. Truth tables of multi-value functions are also available to speed simulation by table look up.

Circuit Type Assignment

The only changes made in the design file by this program will be circuit types for single-

valued functions (for example, cell type or fixed array LSIs with internal interconnect needed). If a circuit type is already specified for an equation, it is checked; if not, a circuit type is assigned. Manual assignments can be made through load equations or on input cards to circuit assignment. Circuits are assigned or checked according to an equation's logic, fan-in and fan-out. Manual assignment is made by specifying a circuit type for a subject symbol on an input card. The subject symbol may contain dashes as in the substitute symbols program to allow assignment of this same circuit type to all equations which have the same subject symbol except for the character positions which contain a dash.

Assignment or checking is done by means of a table in the program which specifies circuit types by logic function, maximum fan-out, and maximum fan-in per term (sum of products form). The table is ordered so that the minimal circuit type will be assigned to each equation.

Circuit Timing Check

After circuit types have been assigned to all equations in the file, this program may be run. The design file will not be modified in any way. Various worst case tests are made through all chains and associated sub-chains under test where a chain is defined as starting and ending at clocked circuits. Turn-on and turn-off circuit delays are accumulated for various tests.

A table contains turn-on and off times for each loading condition of each circuit type for each test condition. Tests are made at various temperatures and for either minimum or maximum delays. Turn-on and off delays are alternated whenever circuits invert pulses. Checking criteria specify maximum and minimum acceptable chain delays as accumulated at the top of each chain. Different such criterion can check chains between different clock phases. Set up, skew and margin are included within these criteria.

Any violations found are listed giving maximum/minimum test at temperature, subject symbol of the circuit at the top of chain, and accumulated turn-off and turn-on delay time. A complete listing of all chains for all test conditions may also be obtained as an option. Errors can be corrected by reasssigning circuit types and/or reloading logic equation corrections.

PHASE III - PLACEMENT OF COMPONENTS

The automatic placement of the selected integrated circuits in the right places on each board or LSI minimizes printed or etched wire lengths. The automatic routing of such wires minimizes cross-over points. Thus the number of LSI or board levels is kept to a minimum reducing to a few hours the many man-years this takes by hand. The elimination of breadboarding and manual layout techniques provides with the advent of large scale integrated circuit technology the only practical method of implementation.

Phase III programs depend upon data processed by Phase I and II programs. Various parts of Phase III programs are applicable only to variable array type LSIs. Other parts are applicable to cell type or fixed array LSIs with internal interconnect wire-routing needed. Note that discretionary interconnect is not covered in the design phases except as various data (for example logic equations) may feed into other programming systems such as Computer Production Assistance and Computer Test Assistance.

Circuit Placement

Manual placement is done by specifying subject symbols and board placement coordinate locations. Possible errors are: 1) Circuit types not in file, 2) Subject not in file, 3) Duplicate subject symbols, 4) Placement location overfilled. Automatic placement will generate board locations for each subject. Circuits within the same flat-pack for fixed array type LSIs or within the same cell for cell type LSIs will be automatically assigned. Thus, placement will update the file with placement locations and placed circuit types giving circuit allocation within cells and flat-packs. Location will also specify any expander gate(s) located relative to the expanded circuits.

Violations will list subjects or board locations. Output options include listings of spares, unplaced signals, placed equations. Main output is placement diagrams showing circuits and subjects at coordinate placement locations by board. Variable array type LSIs will not require any internal placement or circuit allocation.

Generate Pin Groups

After placement this program will compute and add to a design file for each equation record an input pin group matching the logic equation's inputs and an output pin group matching the equation's load list.

Equations may be reordered within a set or the inputs of single equations may be reordered to facilitate pin assignment such as interior connections brought out to common pins. Expander gate pins are automatically assigned. Generate pin

groups will only specify flat pack pins not connectors or feed-throughs. These are either generated by special programs or can be added by the manual pin assignments program (see Manual Pin Assignments).

Several listings are generated: Maintenance listing with all data (equations, input pins, load list, output pins, circuit type, placement location) ordered by subject; assembly point to point wiring lists internal to each LSI; assembly wiring lists external to each LSI.

Manual Pin Assignments

LSI and/or printed board connector pins and two-sided printed board feed-through pins as well as other pins may be entered into the output pin groups by this program which is also able to modify existing input pin groups. This is a general program for use with any design technology while generate pin groups is a more specialized program with many versions. Normally, connector pins will be known even before listing pin groups using only the placement diagrams. Thus, a card deck specifying connector pins may be submitted to this program either before or after running generate pin groups but after placement. Unassigned connectors will be listed as will signals which still require feed-through. Output options are the same as given for the generate pin groups program. Pin designations must not be duplicated. This and other possible errors will be checked.

LSI Board Etch

For cell type or fixed array LSIs with internal interconnect wire-routing needed, this program will provide layer separation for cross over and wire routing layout according to the specific groundrules of the design. This is a special purpose program which will exist in many different versions. For variable array type LSIs it will only be necessary to specify the array mask for the word pairs of a multi-valued function as developed in Phase I. For internal interconnect routing, internal pin groups must have previously been specified. Output is routed wire lists by layer or array connection points.

Printed Board Etch

This program separates board layers indicating where plated-through holes are needed and routes printed circuit board wiring. Its function is similar to that of LSI board etch except that the wire routing is between pins external to the LSIs and different groundrules will be followed.

PHASE IV - PREPARATION OF WIRE-RUN LISTS

Phase IV depends upon data processed by Phases III, II and I programs. Programs of this phase provide the wiring output in various forms for production assembly. Wire-listings are reordered to run a scribing machine to prepare masks for LSIs and printed boards. The massive data transmitted from Engineering to Production is literally 100,000's of wire segments necessitating masks automatically prepared by computer controlled scribing.

LSI Board Wiring Lists

Either a routed wire list or an array mask list is made for each LSI in the file.

Printed Board Wiring Lists

A routed wiring list is made for each board connecting flat-packs.

Mother Board Wiring Lists

A routed wiring list is made for each board connecting other boards.

Listings Ordered for Scribing

Cards, paper-tape or magnetic tape is produced to run a scribing machine.

SERVICE PROGRAMS

These programs are used by the programmer to run and service the computer design assistance system. The library program forms a basic package from which all other programs are built up. The executive provides continuity of running between all the other design programs and communication with the computer operator. The file edit works on design files as a whole copying, deleting or renaming them. In addition, various parameteration programs may exist to modify tables in other programs (logic subroutines in simulation program, circuit types in circuit assignment program, time delays in timing check program, etc.). Service programs are:

Subroutine Library

In addition to a library of logical, input/output editing and formatting, sorting, scanning, searching and square root subroutines, constant and data format pools are specified. All are designed to simplify the programming of the type of programs found in the computer design assistance system. The use of a compiling system simplifies library

maintenance. Programs are constructed by copying common portions from the library. Presently the Cobol Compiler, Update and Library system is being used for Bootstrap development and maintenance of all programs.

Executive

This program provides continuity between all the other programs. The executive consists of two parts: start-up and run. The start-up reads the first control card and checks magnetic tape file mountings. The run executive is then called.

The run executive reads a single program batch control card which preceeds each design program input card deck. Any tape reassignments necessary, rerun points, comments to the operator, etc. are made and the next program to be run is called.

Thus, a batched system card deck is submitted for each run. The run executive calls all other programs which return back to it upon exit. Normally there will be no computer stops for a run. Note that the same program may be run more than once with different data. Also more than one design file may be updated.

File Edits

This program will: 1) copy the complete CDA system of files and change the sequence number; 2) delete selected records of selected files; 3) copy selected file records of selected files; 4) copy selected files with new titles; 5) rename selected file titles; 6) list all file titles; etc.

Parameter Updates

Various special programs may be provided to update tables in other programs.

CONCLUSION

The objective has been to show various computer design aid tools applicable to multiprocessor LSI technology and to provide a cooperative man-machine interactive design system to free the designer from routine bookkeeping tasks so that he may devote more of this time to the actual system design.

Since all programs work on the same file, corrections can be made at any point in the design by simply rerunning appropriate programs. Also the fact that the programs are independent with data separated from the programs into a common file allows for easy expansion of the system by adding of new programs to take advantage of new construction techniques as they come along.

Since each phase is complete and dependent only on the preceding phases, development of programs can proceed in parallel with the actual equipment design using the lead time of completed phases.

ESSENTIAL FEATURES OF ON-LINE SYSTEMS

HARRY D. HUSKEY

Dr. Huskey is a Professor of Mathematics and Electrical Engineering, Department of Mathematics, University of California, Berkeley, California. He received his B. S. degree from Idaho in 1937, and his M. S. and Ph. D degrees (Mathematics) from Ohio State in 1940 and 1943, respectively.

He was an Assistant Mathematics Instructor at Ohio State from 1937-38; an instructor at Ohio State from 1941-43; at the University of Pennsylvania from 1943-46; temporary Principal Science Officer, at the Natural Physics Laboratory, England, 1947-48; with the National Bureau of Standards, Wash., D. C., 1948-49; Assistant Director and Instructor in Numerical Analysis, University of California (Berkeley), 1949-54; Associate Professor, Mathematics and Electrical Engineering, University of California, 1954-58, and was made a full professor in 1958, his present position.

He was Technical Director of the Computer Laboratory, Wayne University, 1952-53. His specialties include the following: design and use of electronic digital computing machines and accessories, mathematical area of surfaces, and the solution of algebraic linear simultaneous equations.

He is a member of AAAS, Mathematical Society; Industrial Mathematics Society; Mathematical Association, the Association for Computing Machinery; IEEE, and the British Computer Society.

N 67-17110

ESSENTIAL FEATURES OF ON-LINE SYSTEMS

By Harry D. Huskey

Professor of Electrical Engineering
Massachusetts Institute of Technology and
University of California, Berkeley

The exciting interest in time-sharing of computers that is sweeping the profession raises some questions. Why, is time-sharing so good? If it is so good, why didn't we do it years ago?

On the first question: time-sharing is not so good! What is good is the on-line use of computers wherein the user works for continuous periods with sustained attention. Thus, relative to batch processing techniques the user can concentrate on his most significant problems. There is no need to have a number of secondary problems active in order to occupy his time while waiting out turn-around on a batch system.

Even though the above is sufficient economic justification, of much greater significance is the fact that the user may "explore" his way through a problem. In other words, the user may start on his problem without knowing a complete solution algorithm. If certain cases never occur they need not be accounted for in the program.

This on-line use contrasted with batch use can be compared with communication with a distant person. Letters represent batching of information and must be sufficiently complete so as to avoid misunderstanding. In comparison a telephone communication gives (1) the possibility of a briefer text assuming the receiver understands the ambiguities or the omissions, (2) if the text is too brief immediate feed-back occurs, and (3) immediate response (answers) can occur. Note that the turn-around time for a telegram compares with that for batch processing computers.

Now consider the second question: The first electronic computers were used on-line. And, even in recent times, groups doing large system programs, where total elapsed time to completion was of prime importance, have been given exclusive on-line access to large computing systems.

However, most computer systems have developed into well-tuned batch systems with input and output queuing organized to keep the central processor busy (this, generally, maximizes the amount of computing that can be done).

A successful on-line system must service a number of users comparable to (or more than) the number that can be serviced with an equivalent (measured, say, in terms of cost) batch system. In order for this to be possible the portion of the system (hardware, software, buffer areas, etc.) dedicated to the individual user must be small. Individual start-up and close-down costs must be modest with respect to the system and to the user. Minimum price terminals are low data-rate devices and this requires system storage of user files. Although, magnetic tape represents a possible means of storing such files between user-sessions, it is much too slow for the storage of the files of a user when he is active.

In the above discussion the consideration of display scopes is being by-passed because of their high cost per user. Technical developments may change this cost. Of perhaps greater importance: with a proper supporting system (yet to come) a scope by pictorial techniques may transmit to the user so much more information than is possible by a typewriter, and pointing techniques (light-pen, Rand tablet, etc.) may be so much easier than, say, typing coordinates; that scope consoles may then compete economically with typewriters for some applications. Summary of the desirable characteristics of an on-line system:

1) The terminal (and all of the system dedicated to the individual user) must be low-cost, and in the current state-of-the-art this implies low data-rate equipment.

2) Low data-rate terminals imply substantial amounts of system storage per user (active or not).

3) The active files of active users must be in quickly accessible memory. A "lively" on-line system must respond to each active user in seconds. For general engineering and scientific computation this means (1) the user's files are in fast core (which may conflict with the minimum cost requirements mentioned above), or (2) the user can be "swapped-in" from a lower-cost memory without excessive overhead costs.

The system of Project Genie at the University of California, Berkeley, represents our approach to the above problems. User terminals are teletype machines. User storage is a large capacity disc (30 million words in the initial version); low cost storage for active users (and system programs) is a million word drum which can swap with the high speed core at core-rate. The 32K memory is divided into two frames and variable priority memory access is provided to allow the central processor to run during drum and I/O transfers. The variable priority reduces memory access conflicts from about 40% to below 10%. When a device (drum, CPU, data channel, etc.) requires a memory cycle, it comes in with minimum priority and for each memory cycle that it fails to get its priority is upgraded.

The hardware configuration is less than half the story. Batch processing software is not suitable for on-line systems because it is not designed to provide interactive communication with the user.

A number of so-called on-line systems have been promoted which either (1) provide limited language capability requiring restricted input and

output formats or (2) which permits the user (via his console) to obtain a position in the batch processing queue. In order to compare at all with batch processing the on-line system must provide essentially the same computing facilities.

The on-line system must cater to at least two types of user. One is the neophyte or occasional user. To him the system should be "forgiving" and it should "lead" him through the computational process. At the user language level, JOSS (our version at Berkeley is called CAL) is an example of a good "fail-safe" language. This tolerance and helpfulness must exist at all levels such as system communication, text generating and editing, and in each user language.

At the other extreme is the professional user. For him flexibility is of prime importance. He should be able to maximize his communication rate with his problem relative to his physical and mental effort. Thus, system procedures should all be available to him with a relatively uniform method of calling.

All the above must occur in a system which gives almost all hardware and software facilities to each of its users (varying perhaps from 20 to 100) for his time-slice out of an interval which is at most a few seconds long.

The Berkeley system uses memory paging to protect programs from one another, and to extend the apparent memory size from the user's viewpoint.

People are realizing, and some computer companies have apparently not yet realized, that the proper software is at least half of the cost of a good batch processing system. This is even more true of software systems.

ON-LINE SIMULATION

MARTIN GREENBERGER

Dr. Greenberger is Associate Professor at the Sloan School of Management, Massachusetts Institute of Technology. An applied mathematician, his interests are in the application of computers and quantitative methods to decision-making and economic behavior. He has been engaged at M. I. T. 's Project MAC in the development and use of interactive computer systems.

Professor Greenberger received A. B. (summa cum laude), A. M., and Ph. D degrees from Harvard University in Applied Mathematics. While at Harvard he was a National Science Foundation Fellow,a Teaching Fellow in Mathematics, and staff member of the Harvard Computation Laboratory.

Before joining the M. I. T. faculty, Professor Greenberger formed and managed Applied Science Cambridge, the IBM group that cooperated with M. I. T. in the establishment and operation of the M. I. T. Computation Center. This group assisted the Smithsonian Astrophysical Observatory in tracking the first Russian and American satellites.

Dr. Greenberger became Assistant Professor at the Sloan School in 1958 and Associate Professor and Head of the Quantitative Section in 1961. In 1962 he edited a book of essays entitled Management and the Computer of the Future.

Professor Greenberger is co-author of the books, Microanalysis of Socio-economic Systems -- A Simulation Study, and On-Line Computation and Simulation: the OPS-3 System. He has written numerous articles for technical journals and magazines. During the 1965-1966 academic year, he was a Guggenheim Fellow at Berkeley.

ON-LINE SIMULATION IN THE OPS SYSTEM

By Martin Greenberger
and Malcolm M. Jones

Associate Professor of Management
and Instructor of Management
Massachusetts Institute of Technology
Cambridge, Massachusetts

SUMMARY

The OPS system, an interactive system designed for use in a time-sharing environment, includes an on-line simulation capability. A simulation activity, thought of as a series of events, is scheduled, canceled, or rescheduled dynamically on the AGENDA, either at a specified time, or when a prescribed condition is met. The activity can be made to consume simulated time by means of an internal delay for a certain period, or a wait until given conditions are satisfied. The AGENDA is a time-ordered list of all conditionally and unconditionally scheduled activities. The user may inspect it at any point in a simulation, and personally modify or restructure it. He may base his strategy on data and partial results examined and analyzed with the help of the OPS system during interruption of the run. Extensive tracing facilities permit the user to follow the flow of control during a simulation to any level of detail.

Introduction

OPS is an interactive system designed for general use in a time-sharing environment.* It includes an on-line capability for building models and running simulations. Simulation activities are scheduled, canceled, or rescheduled dynamically on an AGENDA either at a specified time or when a prescribed condition is met. Activities can be made to consume simulated time by means of a delay for a certain period or a wait until given conditions are satisfied. The AGENDA is a time-ordered list of conditionally and unconditionally scheduled activities.

*For those interested in the origin of names, OPS was originally an acronym for On-Line Process Synthesizer. The system could be adapted for a small stand-alone computer. It is fully documented in the manual On-Line Computation and Simulation: the OPS-3 System, M. Greenberger, M. M. Jones, J. H. Morris, Jr., and D. N. Ness, MIT Press, 1965.

Working within the multi-purpose framework of the OPS system, the user may inspect the AGENDA or some index of performance without stopping the simulation. He can also interrupt the run to make unprogrammed inspections and alterations. Before resuming, he can roll the simulation back to an earlier state that has been preserved, or perturb it in some other manner. Reference to data and activities is symbolic.

Extensive tracing facilities permit the user to follow the flow of control during a simulation to any desired level of detail. He may modify his experimental design as he views partial results, as well as conduct interim statistical analyses, without relinquishing title to the computer or losing his place in the simulation. By running independent components of his model singly or in selected combinations from standard initial conditions, he is able to examine different aspects of his simulation in a controlled way. This flexible mode of operation encourages him to build and validate his model incrementally, thus giving him a measure of protection against the problem of initial overcomplexity that can plague a monolithic simulation.

The OPS System

The OPS system provides a multi-purpose facility for on-line computation, programming, and model-building. It is an open system and it is modular. The user can enlarge and reshape it to suit his own requirements by adding individually tailored subroutines, known as operators. An operator may simply be a subroutine with a fixed number of arguments of fixed connotations. Or it may have a variable number of parameters whose interpretation is sensitive to context. These parameters may be read in literally, symbolically, or with conversion to any of several modes. Some 60 to 70 standard operators come with the system. Additional operators may be written in any of a variety of programming languages, such as FORTRAN, MAD, or FAP; or they may be recruited from a wide assortment of existing subroutines without modification.

New operators may be written in terms of old operators. An ordered set of operators is known as a compound operator or KOP (pronounced K-OP). A KOP may be executed as it is being constructed, since its execution is interpretive. After it has been debugged, it may be compiled into a conventional subroutine. A KOP has a fixed

number or arguments of fixed connotation. Whether or not it is compiled, it is referred to by name as though it were a subroutine. Compound operators (KOP's) may themselves be compounded. They may call themselves and each other to any depth. Flow of control between KOP's is similar to flow of control between subroutines.

There are standard operators for input and output, testing, branching, and repeating within a KOP. Thus, a KOP is analogous to an ordinary program, except that its components can be of arbitrary complexity and tailored to an individual need. Operators are the building blocks and KOP's are the structures.

In the OPS system, all variables are referred to symbolically through a symbol table maintained during execution. Changes in the symbol table can be made at any time without disturbing the definitions of activities. Arrays of up to 3 dimensions can be addressed by implicit indexing. Thus, the multiplication

```
SET X = A * B
```

applies whether X, A and B are single cells or arrays. If A and B are compatible arrays, the multiplication is carried out element-by-element over all of their elements. Infix symbols are available for matrix multiplication (.M.), matrix transposition (.T.) and the differencing of elements of vectors (.D.). If either A or B is a cell, the designated scalar operation is performed.

The current version of OPS does not include a general list processing capability, although several list processing operators have been used experimentally. It also appears feasible to add the SLIP primitives to OPS, subject to core space limitations.

Simulation and Model Building

A simulation model may be constructed from operators and KOP's by using them to represent activities. An activity is an ordered sequence of one or more events, or more precisely, a list of operators defining how these events take place. Since the order of execution of activities is not known in advance, the flow of control between activities is not handled in the normal style of subroutine calls. A special KOP, known as the AGENDA, is introduced to permit the dynamic scheduling of activities. After an activity is completed, control returns to the AGENDA which specifies the next activity to be executed. Activities are scheduled on the AGENDA for execution either at a specified time or <u>when</u> a specified condition is met. Activities can schedule, cancel, and reschedule themselves and other activities during execution. They may also consume simulated time by delaying for a certain period or waiting for a condition to be met. <u>Delays</u> and <u>waits</u> are used to string events together as activities. These features are in the spirit of simulation languages such as SOL and SIMSCRIPT, although SIMSCRIPT works only with events and does not schedule conditionally.

In designing the simulation facility, primary emphasis was placed on providing the researcher with a flexible framework for building his model on-line in an interactive manner. The building phase of the simulation process was considered more critical than the running phase, and the subject of running efficiency received secondary status. This philosophy led to a combined interpretive and compilative system. Operators are compiled programs. They run at full efficiency. KOP's, however, are executed interpretively and may be traced in detail, a feature which is particularly effective in an on-line environment. Once a model is ready for production runs, its running efficiency can be improved by compiling all activities written as KOP's into operators.

In building a simulation model on-line, there is great advantage in structuring it so that preliminary pieces can be tested before they become embedded in a larger whole. This sometimes is best accomplished by building from the outside in, as in the construction of a house. Other times, it can be achieved by assembling parts in hierarchical combinations, as in the formation of an organization. Either way, relatively independent parts should be isolated into separate segments. This allows the computer to aid in weighing alternative formulations of components, and helps build an understanding of the model as the model itself is built.

In the OPS system, the parts of the model are out in the open and easily modified. The AGENDA or schedule of activities also is out in the open. Through the use of system switches, the user can indicate where in the AGENDA he wishes to start the simulation, the exact duration of the run, or a condition for termination. He can insert himself into the simulation, and modify its course from the console by altering the AGENDA or adding to it. He can interrupt a simulation to examine some data, make a calculation, transform a variable, or estimate a coefficient; then insert a change and resume the run. This type of interaction is facilitated by the openness of the OPS system. All the simulation variables are available for examination, and they may be operated on by any of a wide variety of statistical operators. Since all KOP's are executed interpretively, it is straightforward to modify a KOP and then restart the simulation. No intermediate compilation or reloading of the system is required.

Extensive tracing facilities are available. For example, the following may be traced: the names of KOP's executed; the line numbers executed; the parameters and results of operators executed; the movements of simulated time; and the values of any variables referenced symbolically. This tracing is controlled by system switches which may be set at the console or dynamically from within a KOP.

The man-machine interaction possible with this type of on-line simulation facility offers new possibilities for better understanding of complex systems. For example, those aspects of a system that are well understood may be programmed as operators or KOP's. The less understood components of the system may be modelled by the researcher at his console. The entire system may then be made to interact under different controlled conditions. The simulation may be stopped by the user at any point, by pressing a special interrupt button, and detailed validation analysis performed. It is not necessary to specify the desired stop point in advance, although this alternative is available. Alternate simulation strategies may be compared by restarting the simulation from a given point with different decision rules. It is only necessary to edit the appropriate KOP's, reload the AGENDA and common storage (which has been saved on disk), make appropriate modifications to the simulation variables, and enter run mode.

Activities

A simulation system must have a way of representing events or activities. In SIMSCRIPT and SOL this is the subroutine. In OPS, it is an operator or sequence of operators, called a KOP. An activity may affect a simulation by changing the values of state variables in common storage. It may also alter the course of the simulation by scheduling the execution of activities, including itself, at future times, and by canceling activities previously scheduled. An activity may also advance simulated time by means of DELAY and WAIT operators.

The Agenda

A discrete simulation system must also have a scheduler that drives its clock. In SIMSCRIPT, this is the events list. In OPS, it is a special KOP called the AGENDA. The AGENDA normally contains calls to activities. The entries in the AGENDA are ordered by their line numbers which are equivalent to simulated time.

At the top of the AGENDA are conditional calls to activities, calls that depend upon some relation among state variables. Following the conditional calls are unconditional calls. Normally the AGENDA is entered from the top and the conditional calls are examined to see if any of them is satisfied. If one is satisfied, the first unconditional call is executed, and the system variable TIME is advanced to the line number of that call. The variable TIME always contains the current value of the simulated clock. It is not changed when a conditional call is executed.

All the call operators in the AGENDA delete themselves from the AGENDA when they are executed. Thus, an activity is not called more than once unless it is scheduled more than once.

Note: The preceding is the first part of a talk presented at the 21st National meeting of the Association for Computing Machinery, Los Angeles, August 30, 1966. The full text appears in the Proceedings of the conference, published by the Thompson Book Company, Washington, D. C., 1966.