MASSACHUSETTS INSTITUTE OF TECHNOLOGY

APOLLO

GUIDANCE, NAVIGATION AND CONTROL

E-2097

A MULTIPROCESSING STRUCTURE

by

R. L. Alonso, A. L. Hopkins Jr., H. A. Thaler

March 1967

MIT INSTRUMENTATION LABORATORY

CAMBRIDGE 39, MASSACHUSETTS

GUIDANCE, NAVIGATION AND CONTROL

Approved: Eldon C Hall Date: 4/7/67
E.C. HALL, DIR. DIGITAL DEVELOPMENT
APOLLO GUIDANCE AND NAVIGATION PROGRAM

Approved: D. G. Hoag Date: 6 Apr 67
D. G. HOAG, DIRECTOR
APOLLO GUIDANCE AND NAVIGATION PROGRAM

Approved: Ralph R. Ragan Date: 10 Apr 67
R. R. RAGAN, DEPUTY DIRECTOR
INSTRUMENTATION LABORATORY

E-2097

A MULTIPROCESSING STRUCTURE

by

R.L. Alonso, A.L. Hopkins Jr., H.A. Thaler

March 1967

MIT

CAMBRIDGE 39, MASSACHUSETTS

COPY # 93

ACKNOWLEDGMENT

This report was prepared under DSR Project 55-36200, sponsored by the Manned Spacecraft Center of the National Aeronautics and Space Administration through Contract NAS 9-4065.

E-2097

A MULTIPROCESSING STRUCTURE

ABSTRACT

Extrapolation of Apollo experience to spacecraft computers of the next generation indicates a need for digitai 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.

by R. L. Alonso
A. L. Hopkins, Jr.
H. A. Thaler
March 1967

TABLE OF CONTENTS

1. INTRODUCTION

This report is based on a paper[1] by Alonso, Hopkins and Thaler with some minor modifications and additions, mostly in the way of examples. It represents an approach (from among many) to computer organization which seems to hold promise for both reliability and flexibility. Many obvious areas of great importance have not been dealt with, however, and this note is offered more as a stimulant than as a serious, completed proposal.

1.1 Design Trends

In manned spacecarft 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 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.

1.2 Multiprocessors

Extrapolating the Apollo mission to a planetary mission has many pitfalls,

1. Alonso, R. L. , A. L. Hopkins, Jr. and H. A. Thaler, Design Criteria for a Spacecraft Computer, NASA Electronics Research Center, Spaceborne Multiprocessing Seminar, Cambridge, Massachusetts, October 1966.

as entirely new problems and solutions are involved. From the computer's point of view, however, the requirements can be expressed independently of many of the attributes of the total spacecraft. Size and power constraints should not be expected to be much different than 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, A hat is needed i a 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 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. The new concept, 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 besides 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 changes other than the addition of access lines for the new system to the coaxial cable (or waveguide) run. Today, multiwire cable and connector problems probably constitute nalf the battle in making spacecraft systems work.

Another example ot 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 the future we can expect on the order of a hundred programs running at once and tons or hundreds of samples per second per program,

1.3 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 this 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 semi-conductor 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 wihtout 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.

2. IDEALIZED MULTIPROCESSOR STRUCTURE

2.1 System Structure

As a model upon which to base our size and performance estimates we use an organization which is simple, yet 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 Fig. 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 and that the contents of memory address k is desired. Upon receipt of this message, 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

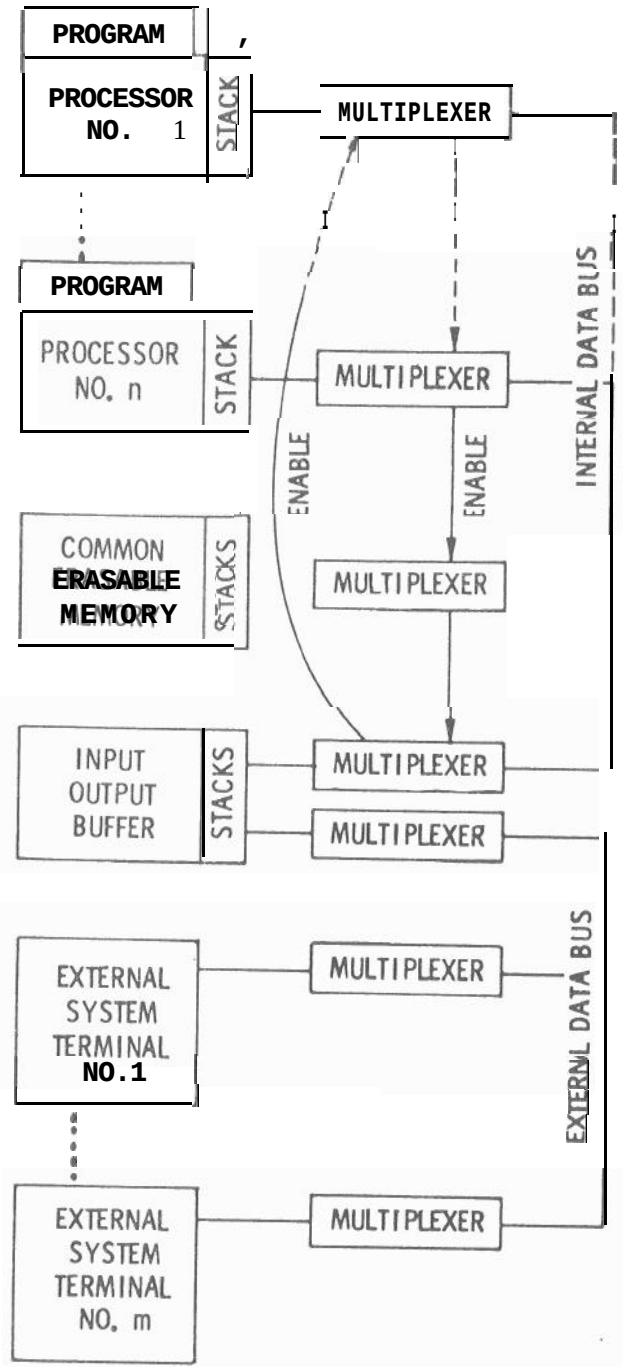

Fig. 1 Collaborative multiprocessor model.

external data transfers on a separate bus system. This not only isolates the mutliprocessor 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.

2.2 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:

2.2.1 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, elc.

2.2.2 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.

2.2.3 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.

3. OPERATION

3. 1 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, + Δ > Y_i
C R	Conditional Request
R (J_i, Y, T)	Job request message
A (J_i, P)	Job acceptance message
E (J_i, P)	End of job message
T	Time - floating point

The general job assignment can be as follows:

1. R (J_i, 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_i which has priority Y_i 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 message R (J_i, Y, T) appears on the bus the free processors P_1, P_2, . . . P_j each compose a response message

A (J_1, 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_1, 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_i 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, I') issued is recorded in every stack, 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. Each stack must have as many extra cells for A messages as there are processors.

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 $T_0 > T$, where T_0 is the present time, and further sorted by priority. For each new T_0 the stack is interrogated to see if one or more jobs arc outstanding. If so, an A-message is prepared, as in 3.

It is sometimes desirable **to initiate** a job as a result of the completion of several previous jobs, whose order of execution is uncertain. For example, it might be desired to initiate J_5 when ail of J_1, J_2 and J_3 have been completed; but J_1, J_2 and J_3 are independent jobs, executed in arbitrary order. There are some potential synchronism problems in this sort of scheduling which can be eliminated by the modification of the joh request message format to include conditional information; for instance, let J_1 issue R (J_5, Y, T, K_1), where K_1 means "condition 1"; similarly, let J_2 issue R (J_5, Y, T, K_2), and J_3 issue R (J_5, Y, T, K_3); if J_1 issues its R message first, R (J_5, Y, T, K_1) will be in all stacks. No processor accepts J_5 because only one of the three necessary condition has been met thus far. Suppose that J_3 then issues R (J_5, Y, T, K_3), and let this message be merged with the previous conditional request, so that the stacks will then hold R (J_5, Y, T, K_1, K_3). When R (J_5, Y, T, K_2) is eventually issued, the merging process is repeated and the stacks hold R $(J_5, Y, T, K_1, K_2, K_3)$. The stacks can be made so that all three conditions must be present for a processor to accept that job.

Unconditional job requests can be made by issuing messages R $(J_1, Y, T, K_1, K_2, K_3)$, i. e. , by simply fulfilling all necessary conditions with one message.

3.2 Job Stack

The stack associated with each processor 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 titne. 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 consistent with a time is 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.

3.3 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_1, 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.

4. IMPLICATIONS

4.1 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 from 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 undoubtly result in inefficient processor usage, but it might lead to useful segmentation techniques.

4.2 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 fcr 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 space-craft data processors in a decade that there are on the ground today.

4.3 Example of Job Assignments

Figure 2 shows the various states, as time passes, of a system composed of three processors and a common memory. The heavy line is the Internal Data Bus, and the arrows to and from the processors or common memory indicate which paths are active at that time. For clarity, only the stack associated with each processor is shown, and not the processors themselves. The single cell to the left of the stack is a transmitter; outgoing messages are placed there for issuance to the Internal Data Bus at the proper time. The group of three cells represents the associative stack,

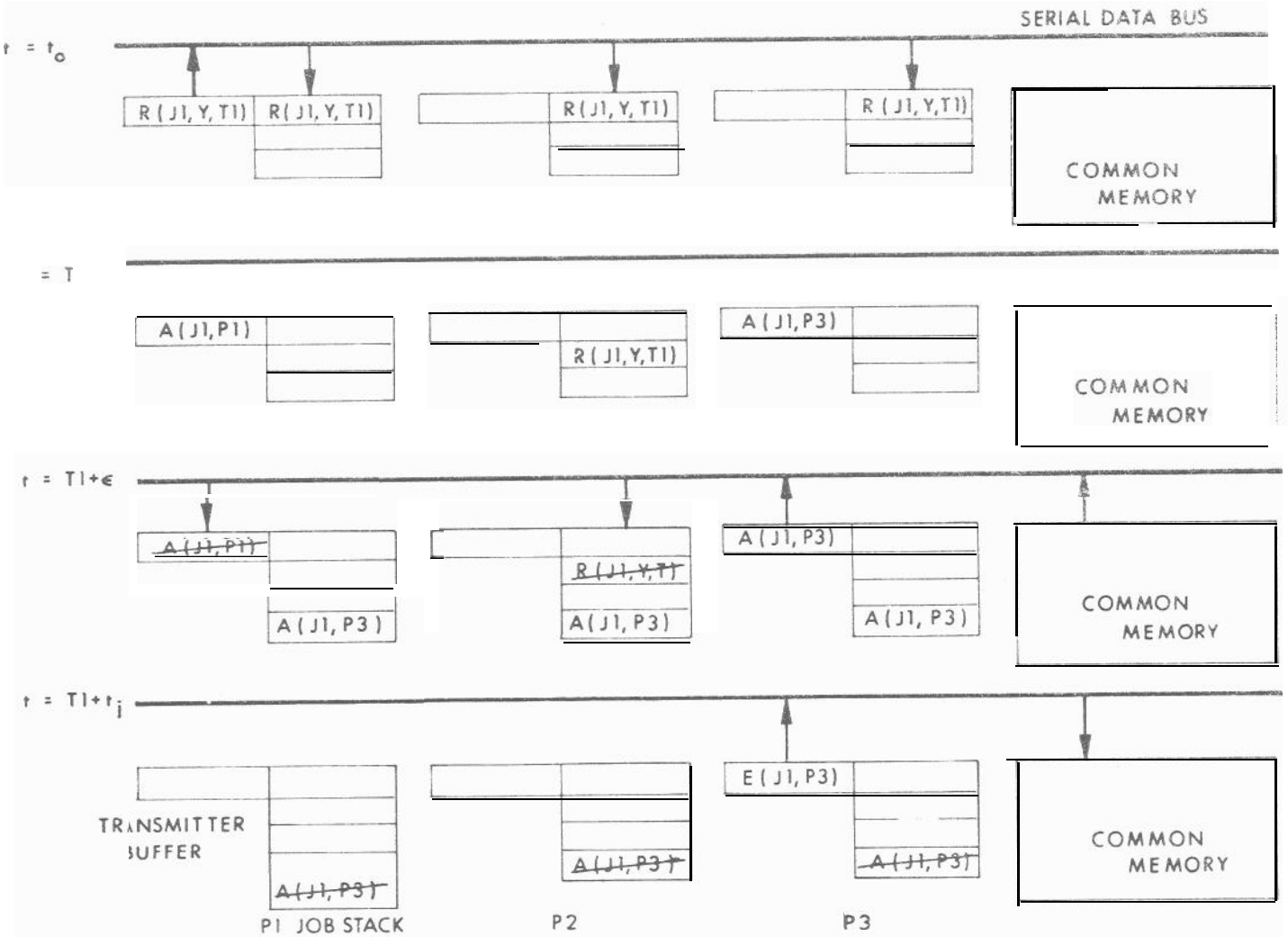

Fig. 2 Job example

At time T = to processor P_1 issues a request that job J_1, or priority Y, be executed at time T_1. All three processors, including P_1, record the request in their stacks.

When time T_1 arrives, two processors, P_1 and P_3 find themselves in a position to accept the Job. Both prepare acceptance messages A (J_1, P_1) and A (J_1, P_3) and place them in their respective transmission cells, waiting for their turn at the bus. In our example P_3 happens to be the first to gain access to the bus, and hence the message that appears on the bus is A (J_1, P,). Processor P_1, upon receipt of A (J_1, P_3), at time T_1 + a both records the message and elides its own waiting A (J_1, P_1). In fact, all processors record A (J_1, P_3), and all other recorded messages which have J_1 in them are deleted. Processor P_2 still had the request R (J_1, Y, T_1) in its stack at t = T_1, but that message is deleted and A (J_1, P,) recorded.

This system avoids the problem of multiple requests for the same job. Notice also that every processor has within its stack information as to which jobs the other processors are doing.

After P_3 accepts J_1, it obtains from common memory all the pertinent data. Since each processor is presumed to have within it all programs, each processor also has, associated with each Job name, a list of the necessary information to be obtained from common memory.

Some time later, at T1 + tj, processor P_3 finishes its calculations and sends to the appropriate place in common memory the results of its calculations. After this step (or as part of the same message) P_3 issues an end of job message E (J_1, P_3) which deletes from all the stacks the acceptance messages A (J_1, P_3). Part of J_1 could have been to issue other job requests, or even to issue a new Job request R (J_1, Y, T_2) for performing J_1 again at time T2. There is no problem with E (J_1, P_3) deleting a possible R (J_1, Y, T_2) which might have been issued during the course of J_1 because E messages can only delete A messages.

4. 4 Failure Processing

Suppose P_3 had failed in the middle of J_1. Under our assumptions any failure within a processor will be detected before the end of its current job, and before issuing false results. Let failure of a processor result in the issuing of a failure message F (P_i), or F (P_3) in our example. Then all the acceptance messages with P_3 in them, which are stored in all the stacks, can be reverted to R messages, so that failure of a job in progress has the same result as reissuing the original job request. The worth of this procedure is naturally highly dependent on error detection.

5. COMMON ERASABLE MEMORY ORGANIZATION

In the foregoing analysis it was assumed that the common erasable memory was an infallible monolithic structure. In a real common memory design it will be necessary to employ some form of redundancy to meet the reliability requirement. To simultaneously provide high reliability and bandwidth, the graceful degradation concept may be used.

As an example of gracefully degrading (and expanding) memory structure consider a system in which several electrically independent memories communicate with the message bus via a multiplexer and a data interface unit. The latter contains three lists, or stacks, which are used respectively for paging (like a table of contents), data requests, and output data.

Each memory is logically divided into groups of words called pages, which would probably be from 2^6 to 2^{10} words depending on the designer's choice One of the functions of the data interface unit is to assign physical pages to logical addresses upon command from a processor via a job request. There is no a priori restriction on how many memories shall assign a physical page to a single address block for the sake of redundancy. The paging stack is similar to a job stack in a processor, responding to every access request with an indication of whether or not the memory unit contains the referenced data, and its physical address if so.

Also contained in the data interface unit is a data request stack, similar to a job stack, which buffers data access or storage requests until they can be honored. Data to be stored is held in this stack along with the address. Data which has been accessed is passed along to a third stack where all output data is buffered and fed to the message bus multiplexer for transmission along to a third stack where all output data is buffered and fed to the message bus multiplexer for transmission along with its address and other appropriate identification to the requesting processor.

This scheme allows simultaneous operation of different memories to the extent that requested accesses are distributed among them. It is a graceful expansion and degradation system because there is no interaction between memories save by the message bus. This interaction would occur when accessed

data is redundantly stored in more than one memory unit. When one unit succeeds in delivering the data to the bus, the other memory units trying to do so would be retired from doing it by an annihilation of the request from either the data request stack or from the data output stack.

It is perhaps, noteworthy that logically the individual parts of common memory may be considered to be processors, although likely specialized ones. These memory processors are capable of storing and delivering messages composed of strings of words to be used by the other processors. To achieve redundancy, for example, one or more of these can record a message that appears on the bus. Redundantly recorded messages can be delivered by a memory processor chosen in the same way as the more general kind.

E-2097

DISTRIBUTION LIST

Internal

M. Adams (MIT/GAEC)
W. Aldrich
J. Alekshun
R. Alonso
R. Battin
P. Bowditch/F. Siraco
G. Cherry
N. Cluett
E. Copps
R. Crisp
J. Dahlen
J. DeLisle
J. B. Feldman
P. Felleman
S. Felix
J. Flanders
J. Gilmore
Eldon Hall
T. Hemker (MIT/NAA)
D. Hoag
F. Houston
A. Hopkins
L. B. Johnson
A. Laats
A. Lapointe (25)
L. Larson
S. Laquideira (MIT/FOD)
T. M. Lawton (MIT/MSC)
D. Lickly
L. Martinage
G. Mayo
R. McKern
James Miller
John Miller
J. Nevins
J. Nugent
R. Ragan
G. Schmidt
R. Scholten
N. Sears
J. Shillingford
G. Silver (MIT/KSC)
W. Stameris
M. Trageser
R. Weatherbee
R. Woodbury
Apollo Library (2)
MIT/IL Library (6)

External:

NASA/ RASP0 (1)
Maj. H. Wheeler (AFSC/MIT) (1)

MSC: (10)

National Aeronautics and Space Administration
Manned Spacecraft Center
Houston, Texas 77058
ATTN: R. Chilton (1)
P. Ebersole (1)
W. Rhine (1)
T. Chambers (1)
G. Xenakis (6)

ERC: (2)

National Aeronautics and Space Administration
Electronics Research Center
Technology Square
Kendall Square
Cambridge. Massachusetts
ATTN: Dr. R.C. Duncan

NASA: (2)

NASA Headquarters
600 Independence Avenue SW
Washington, D.C. 20546
ATTN: J. Kantor (1)
Paul Schrock (1)