C-MOS ARRAY DESIGN TECHNIQUES

Contract NAS 12-2233

SUMC MULTIPROCESSOR SYSTEM STUDY

Contract Modification 7

May 1972

Authors: W.A. Clapp, W.A. Helbig, A.S. Merriam

Prepared for

GEORGE C. MARSHALL SPACE FLIGHT CENTER
NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
MARSHALL SPACE FLIGHT CENTER, ALABAMA 35812

Prepared by

ADVANCED TECHNOLOGY LABORATORIES
GOVERNMENT AND COMMERCIAL SYSTEMS
RCA
CAMDEN, NEW JERSEY 08102

SUMC MULTIPROCESSOR SYSTEM STUDY

By

W.A. Clapp, W.A. Helbig, A.S. Merriam

RCA

FOREWORD

This is the System Definition Report specified in Para. 3.b of Modification 7, Contract NAS 12-2233. Work on this study was performed January through March 1972.

SUMMARY

The current capabilities of LSI techniques for speed and reliability, plus the possibilities of assembling large configurations of LSI logic and storage elements, have demanded the study of multiprocessors and multiprocessing techniques, problems, and potentialities. The design of a highly reliable and powerful multiprocessor system of small physical size, which results from such studies, is of great interest for long-life space missions.

The present study first evaluates three previous systems studies for the Space Ultra-reliable Modular Computer multiprocessing system by IBM, CSC, and Intermetrics, respectively, and then describes a multiprocessing system which originates in the best results of these studies and carries them further both in extent and detail. The proposed system is flexibly configured with up to four central processors, four I/O processors, and 16 main memory units, plus auxiliary memory and peripheral devices. This multiprocessor system features a multilevel interrupt, qualified S/360 compatibility for ground-based generation of programs, virtual memory management of a storage hierarchy through I/O processors, and multiport access to multiple and shared memory units.

In conclusion, the present study is viewed as one in a continuum leading to construction of a multiprocessor system meeting NASA mission objectives; therefore, designation and brief discussion of the most profitable details for simulation and further study have been included.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (Continued)

LIST OF TABLES

Section 1.0

INTRODUCTION

This report documents the work performed on NASA Contract NAS12-2233, modification 7, which has expanded the SUMC/DV system definition to a multisystem configuration operating in a multiprocessor mode.

1.1 BACKGROUND

The SUMC/DV computer design provided a viable, real-time computer design concept for a standalone uniprocessor. For several years considerable effort has been directed to devising schemes for employing more than one computer to solve a given problem or set of problems. Multicomputers have been successfully applied to problems that were separable, but not all problems are separable. Distributed and parallel machines like Illiac IV also employ multiple computer-like elements with central control, but again the class of applicable problems is limited. All of the above systems have their limitations in both types of problems to be solved and in utilization of hardware present.

Tradeoffs exist among throughput, hardware, and class of problems intended. Multiprocessors sharing memory banks show promise of a reasonable and cost-effective balance for a space station environment. A multiprocessor system incorporates multiple hardware modules capable of independent operation, with each type of module devoted to performing a particular portion of the overall process required to run a program. Active modules are central processing units (CPUs), input-output units (IOUs) and main memory units (MMUs). An additional feature of a multiprocessor system is that multiple copies of each module type operate relatively independently of one another with no restriction imposed on which modules must share workload with which other ones.

The SUMC multiprocessor system described in this document is such a system. It incorporates the real-time features required for space missions, provides for independent operation for each of the major system modules for high throughput and increased reliability, and achieves a high degree of compatibility with current third-generation IBM S/360 type systems in order to take advantage of software inheritance.

1.2 SYSTEM STUDY GUIDELINES

The major guidelines employed were to produce a system capable of meeting the processing loads projected for typical future space missions like Space Shuttle and Space Station. Previous NASA reports * were used to extract approximate sizing requirements for both the computational load and the supporting data base. Although these estimates varied widely, they provided usable guidelines.

The SUMC multiprocessor system described here meets computational goals. Although detailed timing was not estimated in this contract, RCA is confident that this multiprocessor design, when implemented in state-of-the-art technology, is fully adequate to meet the goals and provide a growth capability.

During the study, the following guidelines were added to these computational and data base requirements guidelines.

- Improve upon third-generation computer systems to provide for real-time applications
- Capitalize on new technology developments as they occur.
- Provide hardware functions to minimize software complexity.
- Provide a fail-safe capability.
- Employ common modules capable of being assembled into several system organizations (simplex, master-slave, federated, etc.)
- Build on the work performed for NASA by IBM, Intermetrics and CSC.

* See Refs. 9, 10, 15, 16.

Alternative concepts advanced in previous studies have been evaluated in Sec. 2 following. The selected multiprocessor design which in the opinion of RCA best meets the above guidelines is presented in Sec. 3. Section 4 suggests the manner of operation of the selected SUMC multiprocessor system by showing implementation of certain operational details.

1.3 SYSTEM TERMINOLOGY

In this report, the attempt has been to use simple, self-explanatory, or conventional terms to designate concepts and physical parts of the multiprocessors under discussion. Description and context within individual sections should make the terminology clear; nevertheless, a few of the basic and frequently used words are introduced in the following overview.

Some confusion arises from the concept of simultaneity which, in computer systems, covers time multiplexing as well as truly parallel activity. Thus multiprogramming is the simultaneous (often time multiplexed) performance of programs as multiprocessing is the simultaneous execution of elements of these programs. Multiprocessing and multiprogramming can be done with a single digital processor with appropriate scheduling algorithms. A multiprocessor, however, is designed for parallel processing, or true simultaneity. To realize its potential, it must be multiprogrammed; single-thread processing is either a highly redundant or else a degraded mode of operation.

The systems studied are multiprocessor systems composed of autonomous hardware processing units incorporating software and communication facilities required to employ this hardware in meaningful and reliable activity over the mission life. Units of the system are the CPUs (central processors), IOUs (I/O units), and MMUs (main memories). The autonomy attributed to units of the selected multiprocessor system conveys an enlarged meaning to that of the usual independence. Not only can units execute their programs in parallel from main memory, barring

interference, but they can execute sequences of executive code for housekeeping, emergency, and recovery.

The multiprocessor system can be configured to operate in several modes. The most usual and that promoting highest throughput is the democratic multiprocessor mode, in which each processor shares an equal opportunity to work at the available tasks of executive and user jobs. Another mode, which has received attention because of its potential for reliability is the TMR (triple modular redundant) mode. In this mode, three like units operate in synchronous parallel with "votes" being taken at diverse times and places within the system as to agreement of status or results.

Activities and effects which involve a unit alone, or two units during mutually dedicated operation, will be called local while those which would involve the whole system (or any member without preference) will be called global. This distinction proves useful in the discussion of control, interrupts, and recoveries.

The software for multiprocessor systems will be modular, with the primary distinction being between collections of routines called in connection with the multiprocessor operating system (MPOS), or executive programs, and the applications, or user programs. Both user and executive programs are entered into the system as jobs which are executed in pieces called tasks which are indivisible and individually indentified by task ID so that they may be accounted for independently of the hardware used to process them.

The memory system is physically backed by an auxiliary memory and is logically dealt with as a virtual memory. Each task sees such a memory as potentially of a size that the sum of what is seen by all tasks would be many times the capacity of the actual physical memory, a mirage that works because the operating system can manage allocations so that tasks without restriction are satisfied within the memory available. Management of stored data requires moving it in units of fixed size, pages, or of variable size, segments. Since the main memory units are connectively located

between units, they are the means of interunit communication with message areas called in this report mailboxes for the exchange of control data.

Real-time interrupt in processing units allows fast task switching to respond to emergencies and failures according to established priority levels. Masked interrupts, also ordered by priority level, allow for flexible task switching to meet the needs of varying jobs. The latter interrupt may be received at any time, but its recognition is under control of operating routines.

Section 2.0

SYSTEM DESIGN ALTERNATIVES

This section discusses alternative methods of implementing various key concepts in a multiprocessor computer design. From the basic discussions techniques for implementing these concepts were selected for a multiprocessor design which is discussed in detail in Sec. 3.0.

Frequent reference is made to previous studies performed for NASA/MSFC by other contractors. In particular, the following material was used:

- IBM Modular Space Station Computer Study
- Intermetrics Multiprocessor Computer System Study Final Report
- Computer Sciences Corporation SUMC Multiprocessor Configuration Control Analysis and Specification

For simplicity in this section when reference is made to specific concepts contained in these documents the author's name is used to denote the document.

2.1 RECONFIGURATION CONTROL

Two important reasons for the use of a multiprocessor are to provide computational capability greater than that available from a single processor and to provide computational capability of some magnitude even under failure of one or more system units. For graceful degradation each unit (CPU, IOU, MMU, etc.) must have two independently defined states, or groups of states: operational and nonoperational.

The IBM study has stated that three necessary components added to the system endow it with reconfiguration capability. These are:

- Address Translation Unit (ATU)
- Communication Control
- State Control

The ATU translates virtual addresses into the physical memory addresses. The Communication Control defines a unit's ability to transfer information between itself and certain other units in the system. CPUs can communicate with all other units in the system, but IOUs and MMUs can communicate only between units of different types. The State Control governs the type of operations, if any, the unit will be performing at any particular time. In the IBM system all units are in one of four states:

- Off-Line
- Nonoperating
- Operating
- Reconfiguration.

Not all states, however, are permitted for all units.

With regard to these three components, the questions to be considered are:

- What characteristics should these components possess?
- Are there any other components whose incorporation into the system would be an asset?
- How do configured components interact?

2.1.1 Address Translation Unit

Of the two studies that discuss the use of address translation for changing virtual address to physical memory address, Intermetrics implements this by means of paging some program into main memory with remainder paged into auxiliary memory. Intermetrics does not discuss the actual mechanism to be used to perform the necessary address translation.

The IBM study describes the mechanism which it recommends. The word capacity of one page is designed to be equal to that of a main memory unit; and the translation points to an address in any one of the MMUs under the direction of system control information.

From the work performed by Denning* and others it can be surmised that optimum page size should be small, of the order of 64 to 256 words. Any implementation must provide address translation on address increments considerably smaller than the size of a main memory unit.

2.1.2 Communication Control

IBM recommends use of a Communication/State Register (CSR) in controlling transfer of data between units. Under software control, the contents of the CSR may be changed during running of a program. Communication control is employed a) to permit isolation of processors from the system, b) to provide redundancy in the control of communication of processors and I/O controllers with memory, and c) to control the assignment of an I/O controller to a processor.

Communication control is not only redundant but appears highly susceptible to single-mode failures and system lockup. If the CSR for an MMU were to become set to inhibit communication with any other unit, then without overriding connection, recovery could not occur except by system reset such as might occur only in power shutdown.

Neither the Intermetrics system nor the system envisioned by CSC uses such a control. It appears that both of these studies have assumed that the communication control will be adequately provided by the ATU and the executive program.

2.1.3 State Control

IBM suggests a mechanization of state control such that a CPU can enter a state which permits it to control the state of all other units in the system. To monitor the operation of this supervisory CPU, a pair of Reconfiguration State Monitor Units (RSMUs) are included. The operator and the CPU in turn monitor the operation of the RSMUs.

---

* Denning, Ref. 6, pp. 153-190.

The CSC system envisions seven different states, one of which is a state to allow the system to run with three like units in a Triple Modular Redundant (TMR) mode. The other states provided in the CSC system could be considered subdivisions of the IBM off-line state. One is a self-test state; the other two make distinctions in reasons for entering into the isolated state.

These distinctions make the system more capable of monitoring the operational status of a unit. However, this capability assumes the presence of a state monitor called the State Control Unit (SCU), which must be of redundant construction for reliability. CSC uses one SCU for state control of all units.

Neither of these systems has included a workable system initialization. When in the initial state the unit (CPU/IOU) must bootstrap itself into an operating state even though its program status registers and address translation registers are initially cleared and no one particular main memory unit is active.

When the changing of the state of one unit must be done by another unit instead of by itself, both systems end up with a critical link which must be made redundant. However, the system may be reconfigured in the event of failure. If, on the other hand, each unit could control only its own state, a hardware-redundant state control (or monitor) unit would not be needed. The monitoring could be done via the system executive which all processors can execute. Furthermore, any failed unit which cannot change its own state into the isolated state for test and repair can be powered down and disconnected by the system operator.

2.2 UNIT INTERCONNECTION TECHNIQUES

The Intermetrics study report describes four general configurations for a multiprocessor which, depending on the particular arrangement used, requires the use of one or more interconnection techniques. For these proposed arrangements of the

system and for other systems which have been proposed and/or built, three design approaches to the interconnection of units have been used:

- Multiple direct connection
- Crossbar switch matrix
- Data bus multiplexing.

The multiple direct connection system was the most commonly used primarily because the technology of the day prevented construction of circuits that could work in any other arrangement. The only driving circuits available were those that need be connected to one end of a transmission line with the single receiver connected to the other. The connection of either a multiplicity of receivers at one end or distributed along the transmission line resulted in excessive signal reflection, creating an unacceptably noisy environment.

The crossbar matrix switch* (Fig. 2-1) was developed as another approach to circumventing these circuit problems while obtaining higher connectivity. Although adding switching elements in the data path, the approach still allowed use of simplified circuit designs.

Most recently developments have been made in circuits to allow the use of various forms of a data bus for interconnection of multiple units. Initially, a single source multiple receiver system was incorporated. Now data bus constructions** permit

* This system is used by Univac, Burroughs and others. A system connecting many CPUs and many peripheral equipments is at least in the planning stages at the Naval Electronics Laboratory in San Diego.

** Many variations are possible, as described in Sec. 3.1 of the Intermetrics report. Many miniprocessors, such as the PDP 11 and MAC-16, use this type of interconnection.

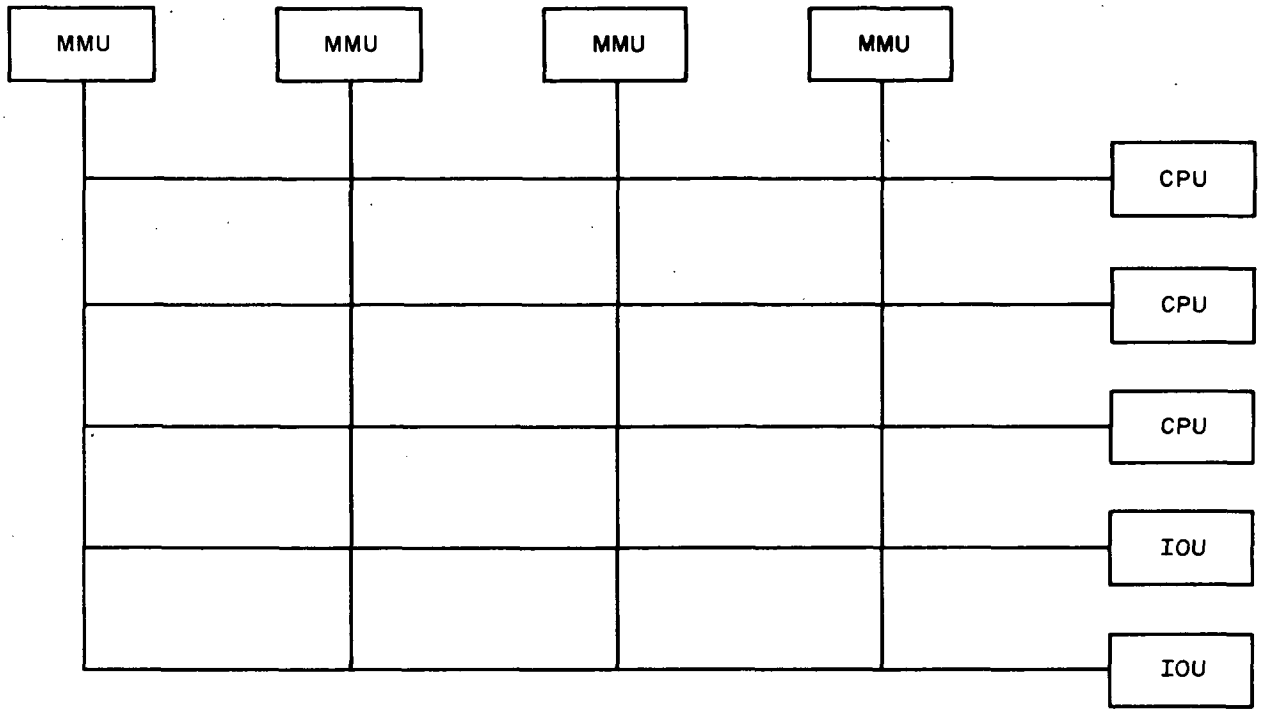


Fig. 2-1. Crossbar switch matrix.

multiple drivers, as well as multiple receivers, to be used for data transfer over a single conductor through the use of driver circuits whose output circuit puts only a high impedance load on the bus when the driver is turned off and the use of high input impedances in the receiver circuits.

To make a choice the salient characteristics of each of these three approaches must be evaluated. There are many variations of each that capitalize on advantages and that circumvent problems of the projected physical medium. The following discussion of the needed designs will emphasize approaches favoring silicon gate C-MOS LSI technology.

2.2.1 Salient Characteristics

2.2.1.1 Multiple Direct Connection

Use of a single driver sending data over a single path to a single receiver characterizes this approach. Currently this technique would seem an inefficient implementation. To provide a one-word parallel transfer path for two-way communication of data between two units requires a total number of conductors greater than twice the size of the data word, extra lines being needed for control. Adding another unit to the system and providing full-duplex data transfers between all units requires twice as many interconnections as before. The number of interconnections and conductors increases in each unit as the number of units in the system grows. Equipment sizes grow and power requirements increase when another unit is added to the system. As the number of signal transfer paths expands, so also does the level of signal crosstalk.

The use of simplex interconnections, however, has some points in its favor. It is an arrangement which incurs the lowest technological risk. The circuits to be used for data transfer require no exacting control of the impedance presented to the conduction path, but they do have to accommodate any offset in reference voltage at opposite ends of the cable. This, however, appears to be no significant problem for a transmission gate driver circuit.

A further point is that the designer of simplex interconnections has the greatest freedom of choice. Data-channel-to-data-channel interaction is nil. Within a given channel the designer may arrange any sequence of timing for the signal without concern for conflict.

2.2.1.2 Crossbar Switch Matrix

A way to overcome the limitations on numbers of senders, receivers and conductors is to provide switches establishing the path. These switches would permit

any one of a number of similar units to be connected to a single driver. If more than one sender unit is in the system, with each requiring unique connection to one of several receiving units, a crossbar switching matrix system, as shown in Fig. 2-1, will meet their needs.

Because of the circuit limitation only one sender may be connected to one receiver at a time. Since any connection made in this system is temporary, dynamic control of the matrix must be provided. Such an operation is usually provided by a switch matrix controller.

Problems characteristic of such an arrangement are:

- Where to put the switch matrix to allow the controller to function efficiently,
- How to prevent a data transfer rate slowdown due to the added distance of sending data to the switch unit and then out to a receiving unit,
- What means are used to prevent crosstalk in the switch unit,
- How to accommodate the multiple differences of reference voltage in several transmitting units,
- How to service a unit or part of the switch matrix without having to deactivate the system, and
- How to construct the switch matrix and its controller so that there are no single-mode system failure points.

2.2.1.3 Data Bus Multiplexing

Present technology allows driving several receivers from a single transmitter, or driving a single receiver from several transmitters. Interconnection of multiport units* is shown in Fig. 2-2. The need for a switch matrix is obviated by placing in each unit a logic network that resolves the conflicts of simultaneous requests for communication.

* This approach is proposed by IBM for use in the Modular Space Station Computer System, Ref. 15, Sec. 2.1.3.

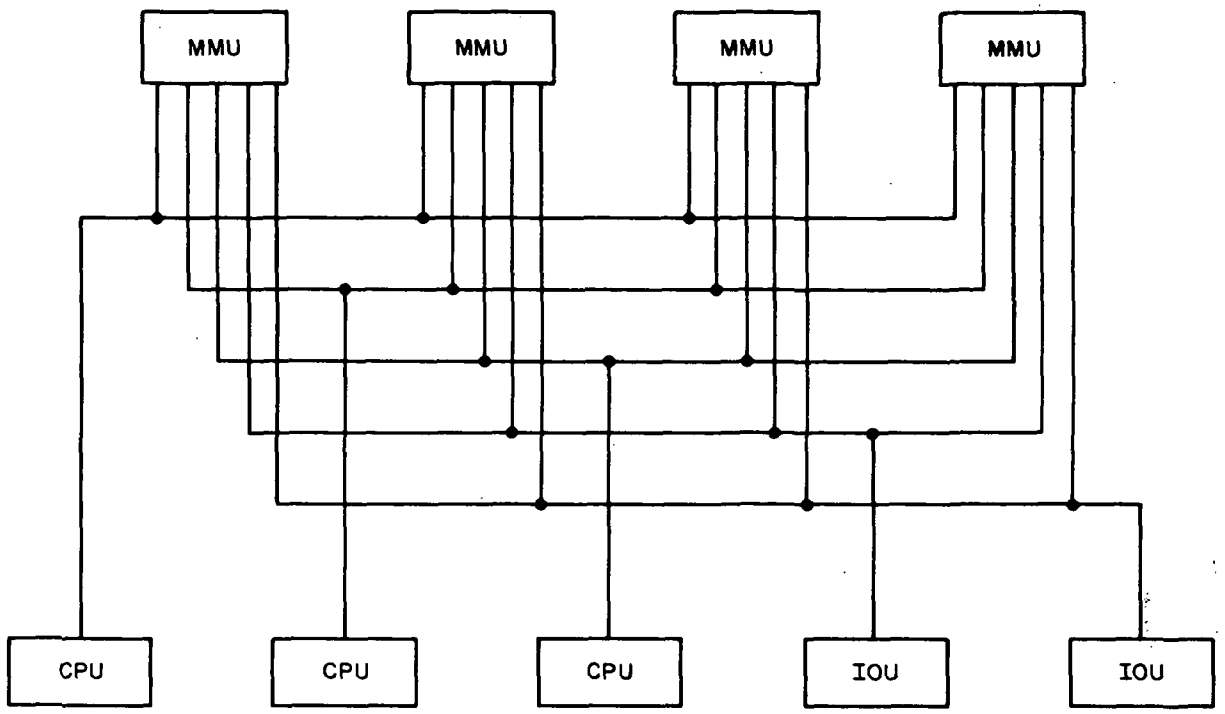


Fig. 2-2. Multiport units.

The number of units of one type or other can be expanded to a maximum size originally planned for the system. A complete redesign of some units is required when the expansion goes beyond that limit. If the capacity provided for a rather large limit, either the unused capacity or the total number of connections required would be extravagant.

Many variations of single conductor multiplexing have developed, e.g., serial transmission and time multiplexing. Time multiplexing needs one time slot for each unit of "simultaneous" data transfer in the fully expanded system. The maximum operating rate of the bus driver and receiver circuits limits the maximum number of "simultaneous" data transfers possible in any one particular system.

The search for a method of going beyond this limit has led to frequency multiplexing techniques*, used so effectively by both the telephone and the broadcast industry. Frequency multiplexing permits multiple channels of high data rate communication to occupy simultaneously the same conductor. Tuned bandpass filters permit data transfers without confusion.

The tuned filter method is technically feasible, but the practical problems of its implementation have previously been overwhelming. The means for modulating or demodulating a carrier frequency to provide a high data rate required expensive, bulky and complex circuits. Provision of a number of these modulator and demodulator circuits for each unit of the system to make all of the units identical in design resulted in a greatly increased size and cost for the units. An approach which uses the previously employed circuits is even more disproportionate in size and cost when applied to LSI units. This approach calls for new modulation/demodulation techniques, of which semiconductor microsonics offers the most compatibility with LSI techniques.

Once a data bus approach is to be used, the choices for design and implementation are numerous. Elimination of many of the construction problems still leaves an almost overwhelming number of organizational choices. For example, a carrier for data transmission and transformer coupling of the circuits to the bus eliminates problems of reference offset voltage and bus system single-mode failures.

The data bus system provides a great potential expansion means for the system. It reduces making connections to units that, as in use of hybrid packaging of LSI arrays, are physically small. Small size makes the data bus system more practical by reducing overall conductor length and thereby removing many of the problems associated with signal transit time.

*Miller, Ref. 16, Sec. 5.6.2.

2.2.2 Interunit Communication Techniques

There are many organizational possibilities for the Data Interface Unit (DIU). Additionally there are a multitude of choices available for coordinating transmission of data over the bus, e.g., asynchronously or through synchronized operation of all DIUs.

The selected techniques will depend on the interconnection system used. For multiple direct connection and crossbar switch matrix no interunit conflicts arise. For data bus multiplexing the use of asynchronous communication can be confusing unless each channel can appear as a single bus. Then the design approach can be the same as for the multiple direct connection system.

Asynchronous communication between units has been used in many configurations. Presently used standard interfaces often employ "handshaking" techniques. The sending unit places the data on the data lines, then sends a control signal on saying that the data is ready. The receiving unit, upon accepting the data, returns a control signal saying that the data has been accepted. Other signals may be exchanged for error control, interrupt and status reporting.

Communication lines between remote units use a completely serial approach of asynchronously sending. To signify that data is ready, the transmitter sends a standard message header. This usually is a single "one" bit. The data bits then are sent in a set of fixed size with standard timing of each bit. When the data word plus error control bits is sent, the transmitter makes certain that no "one" bit appears on the line for a fixed period of time so that the receiver will unambiguously recognize the initial "one" bit preceding a data word.

Systems that employ a combination of serial-parallel techniques, where each part of the data word is sent serially over a wire, have been used only to a limited extent. Varying delays cause the data bits to arrive skewed in time. Previously correction of skew cost a prohibitive amount of hardware. Present design and

construction methods make the added cost less significant. Capability of providing high throughput and excellent error control now make this approach an excellent choice. A specific example of an application of this approach appears in Sec. 3.2.4.3.

Frequency multiplexing to provide multiple asynchronous channels presents challenging organizational requirements when considered for use on the internal data bus (between the CPUs/IOUs and MMUs). The first requirement is for a coordination technique when the number of channels is less than the number of potentially simultaneous users. Frequency multiplexing requires DIUs to request the use of one of the bus channels by notifying the other DIUs that a particular channel is to be occupied for the next period for the transfer of a data word. To avoid disruption of another unit in communication the DIUs must refrain from requesting a previously reserved channel. Any DIU using a channel must also temporarily relinquish the use of the channel between transfer periods to avoid system hangups. Most importantly a DIU must be capable of resolving, with the cooperation of like DIUs and without the help of an outside unit, any simultaneous requests for use of a single channel.

The second requirement is the inclusion of a feature that allows sending the priority code to the addressed MMU before beginning the normal data exchange cycle. Until any access conflicts have been resolved at the MMU, a DIU must not start its data transfer cycle.

There are, depending on the number of users and the number of simultaneous channels available, several methods to resolve conflicts of multiple DIUs simultaneously requesting use of the bus. If a time multiplexing system were used, then, with enough channels to permit all possible links to be active at once, the approach might be to have the bus available to each link periodically. This forces synchronization of the operations of all units connected to the bus and implies that conflicts for use of a single unit are resolved by delaying the user's operation by one or more periods of transfer.

An example might be the request, by two or more CPUs, for access to a single memory module. The simple discipline of having the memory honor the first request it receives after it becomes available will lock out one or more CPUs once one CPU gains access and uses the memory. Stacking requests from the several CPUs not only adds complexity to the memory unit but decreases system throughput because of waits for a bus transfer period.

Allowing users occupancy of the next available bus data transfer period will decrease the delays at the memory module. To provide for filling next time slots from a queue requires a bus controller to permit use of a particular time slot.

2.2.2 Communication of Interrupts

Data transfers over all of the buses in most systems are controlled by either the CPUs/IOUs; i.e., all requests for transferring information originate from one of these units. The exception is that the peripheral equipments usually are capable of asynchronously initiating a request for service. One exception is an operator's console signal to send a message to the CPU system upon manipulation of the send request control. This action by the operator occurs at a time independent of the CPU program execution. When requests for service are initiated asynchronously with bus activity, a special method for transferring them to the CPU must be provided.

Different methods have been proposed as candidate approaches for handling peripherally generated requests for service, separate into two classes. One class is characterized by transfer of signals entirely asynchronously with operation of the computer system; the other is characterized by transfer only after the peripheral equipment has been polled by the system which searches for a service request.

Implementation methods further subdivide the two classes. Polling, for instance, can be such that software controls the majority of functions required. Alternately, these same functions could be performed under firmware control, or they could be wholly implemented in special purpose hardware.

With the IOU able to execute a rich set of instructions, it could carry out the polling operation by software, the slowest of the approaches, which nevertheless allows configuring a system that can perform the polling operation at a rate that uses the data bus and does not produce excessive delays in the time a peripheral equipment waits for service.

Firmware control of the polling operation, which requires no more special hardware than control by software, permits 100 percent usage of the data bus. Therefore any special purpose hardware design can be considered overly expensive. More generally, the use of polling to sense requests for service from the peripheral equipment is foreseen as satisfactory.

2.3 EXECUTIVE SOFTWARE

2.3.1 Common Operating System for SUMC Missions

The IBM study recommends the development of a common executive routine design for all SUMC space applications. This would substantially reduce redundant development efforts as well as the risk inherent in developing new executives for new applications. The practicality of standard executive routines in a variety of data processing, process control and time-sharing utilities has been proved. Not only does this approach eliminate much redundant development work, but it encourages modification and improvement of the standard executive so that after a period of time it becomes more efficient and more reliable than the first version of any custom-built executive. A common implementation of the common design, so that program implementation errors are also removed as a risk factor, should be encouraged. This recommendation does not mean that the executive routine would never be changed or enhanced; instead, it means that, given a functionally decomposable design, one can substitute new modules for old without having to start from scratch. The better the functional decomposition, the easier it will be to add new functions and change old ones.

This approach should also be applied to other significant software components such as file control routines, language translators (compilers) and job schedulers. The IBM study recognizes well that the executive routine interface is the crucial one. Many changes in I/O and storage device speeds, configurations and interfaces can then be made invisible to the application programs, since the effects of these changes can be absorbed by appropriate, one-time changes in the executive routine. Thus, by providing a standard operating system interface, the cost of transferring applications programs from one SUMC mission to the next is reduced to the cost of compensating for changes in the system configuration for capacity reasons only (e.g., memory size, number of disk drives, number of display terminals).

2.3.2 Executive Routine: Hardware, Firmware or Software

The CSC study proposes that serious consideration be given to a complete "reformulation of concepts" of the multiprocessor executive with a view toward implementing the basic control functions of the executive in digital logic hardware. While the authors made a good case for the technical feasibility of doing this and so achieving significant cost savings in hardware, they failed to note, or estimate, the greatly increased development costs and operational risks of a new, untested technology in the nerve center of the multiprocessor operating system.

The use of read-write control memory for microprogram storage would allow the gradual transfer of executive routine functions from software to firmware in response to the results of actual system performance analysis studies. Embedding today's software macroinstructions and control functions in tomorrow's instruction set is probably the most conservative way to improve system cost/performance (like the acquisition of floating point instructions) and it is least disruptive.

2.3.3 Timeline Scheduling

The principal prescheduling control in the space station environment is the timeline schedule. The CSC report calls attention to the need for a mechanism to allow the station crew to make modifications to the timeline via a special, high-level

language. Of even greater potential is its proposal that the modification of timelines due to schedule changes, overruns, experiment aborts, etc., is an excellent application for the SUMC computer system. The MPOS scheduler has the needed information--it knows what time it is, which experiments or jobs were successfully completed, which were aborted, which were started late, which finished early, etc. Therefore, it is appropriate that automatic rescheduling of timeline and batch jobs should be done as a normal MPOS administrative function. The station crew will furnish inputs to this "rescheduler" and will request changes when the schedule adjustments recommended by the rescheduler are unsatisfactory.

The motivation for this feature is to relieve the crew of the burdensome task of rearranging schedules in which precedence rules control the permissible order of execution of jobs. This type of scheduling constraint can remove many superficial scheduling problems from the realm of hand calculation.

2.3.4 Task Scheduling

The CSC study rightly asserts that no single task scheduling algorithm is likely to be satisfactory for the several different SUMC systems they identify (DMS, Experiment and GNC) or other as yet undefined SUMC missions. These CSC results suggest that one or more task schedulers which exhibit the following characteristics be implemented:

1) The scheduler algorithm is actually formulated of parameters specified initially at system generation, and based on empirical studies of the behavior of the system under the job mixes that are typical for the mission at hand.
2) The task scheduler is self-adjusting; i.e., it is responsive to the resource usage levels in the system.
3) In emergencies the parameters controlling the scheduler can be modified dynamically by the station crew using a "symptom/cure" tuning guide in their SUMC handbook.

This kind of scheduling is the best of the alternatives that the CSC study considers; however, the basic scheduling parameters (start and due-out times) of the system are not accommodated in this strategy. Timeline scheduling of jobs means that jobs have both start times (earliest and latest) and they have due-out times (earliest and latest). Failure to take this vital requirement into account in task scheduling is the biggest oversight in the IBM study. The CSC study recognizes that "deadline" is a parameter of task scheduling, but it is not explicitly connected to the due-out requirements of the timeline.

As do most experienced operating systems developers, the authors of the CSC study overemphasize the efficient use of computer system resources and underemphasize honoring the deadlines imposed by the timeline and the station crew. Efficient, timely and reliable performance of the space shuttle and space station missions must take precedence over the efficient use of computer system resources. This is not meant to encourage "over-buy" or to claim that resource utilization is not related to meeting deadlines, but simply to put first things first.

In this light, predictability of due-out time for tasks and jobs is extremely important to the success of the shuttle and station missions and also to the wellbeing of the station crew. The crew does not need the added surprises caused by an unpredictable, erratic computer system. A "slow" but rhythmic response at an experimenter's display or astronaut's control console is easier on the operator (and therefore more efficient) than a "fast" but uneven response — another reason for avoiding time-slicing as a major parameter of the scheduling algorithm. Time-slicing loads the system artificially but does not, in practice, encourage a steady interaction rate.

The best way to get efficient computer resource utilization is to have a large queue of very low priority, nondeadline, jobs which can be initiated on a best-resource-fit basis whenever the current resource levels permit and which can be suspended whenever resource contention endangers the completion of more important deadline work. It is not unreasonable to expect that certain mission housekeeping tasks can be placed in this category.

2.3.5 File Protection and Sharing

The Intermetrics study proposes a data file system in which data sharing and exclusive use of data are controlled by cooperative interlocks between the user programs accessing the data. This system leaves data file integrity at the mercy of the least thoroughly tested user application program. Centralized control of sharing and locking is not only inherently more reliable, but it places control over sharing and locking where it can be monitored and changed, if necessary, without requiring reprogramming of all of the applications which access the pertinent data.

2.3.6 Deadlock Prevention

The specter of system deadlocks in a multiprogramming, multiprocessing environment has troubled the authors of the CSC, Intermetrics and IBM studies. Deadlocks represent one of the hazards of dynamic resource allocation. The deadlock (or circular wait situation) is illustrated in Fig. 2-3.

Task A	Task B
1) Has exclusive control of resource X and will not give it up. 2) Requests resource Y and will not proceed without it.	1) Has exclusive control over resource Y and will not give it up. 2) Requests resource X and will not proceed without it.
Under these conditions Task A and Task B are deadlocked (X and Y might be tape drives, displays or data records). Under this condition neither task is able to progress.	

Fig. 2-3. Circular deadlock.

Certain types of deadlocks can be easily prevented by controlling the initiation of tasks which can potentially deadlock each other. For example, two tasks, each of which contains requests for exclusive use of 10 K bytes of memory, should not be initiated when less than 20 K bytes remain to be allocated. Thus the prevention of

deadlocks resulting from insufficient amounts of a bulk resource can be avoided by: a) supplying sufficient amounts of that resource to support worst-case requests adequately and/or b) refusing to schedule a new task when its potential resource needs could deadlock the system. None of these studies proposed a deadlock prevention strategy in sufficient detail to determine whether this simple form of deadlock could be prevented.

The more serious form of deadlock occurs when unique, nonbulk resources are involved, as illustrated in Fig. 2-4.

Task A	Task B
1) Holds record X in File F. 2) Requests record Y in File F.	1) Holds record Y in File F. 2) Requests record X in File F.
Task A and Task B are deadlocked until A releases record X or B releases record Y.	

Fig. 2-4. Unique resource deadlock.

Deadlock over unique resources turns into single task lockout when all tasks, save one, relinquish their exclusive holds on the resource. In Fig. 2-5, if Task A releases its hold on record X, Task B's request for record X can be satisfied. However, that does not help Task A, since Task A still cannot proceed until Task B releases record Y. If Task B terminates without releasing record Y, goes into an endless loop, or omits unlocking of record Y, then Task A will be stymied until someone releases record Y.

Task A	Task B
t = 1: Holds record X, File F t = 2: Releases record X, File F t = 3: t = 4: Requests record Y, File F	t = 1: Holds record Y, File F t = 3: Requests record X, File F
Tasks A and B are not deadlocked at t = 4, but Task A cannot proceed until Task B releases record Y.	

Fig. 2-5. Single task lockout.

Deadlocks over exclusive use of unique resources are much more difficult to prevent. Ground rules specifying that such resources are to be released within some reasonable period may reduce delay to acceptable proportions, but the problem of overdue library books represents an example of how ineffective this type of control really is. Therefore, in a multiprocessor environment it becomes important to limit the number of exclusively used unique resources and to provide a mechanism for breaking system deadlocks and single task lockouts when the delays become inordinately long.

2.3.7 Interrupt Servicing

The CSC study proposed a trap (interrupt) mechanism for a multiprocessor configuration in which, at any given time, only one processor is designated as an interrupt processor. In other words only one processor (usually a "stopped" processor) is interruptible, putting the burden of responding to all interrupts on one processor. The consequences of this dependency under peak loading of the system could be devastating. Moreover, dedicated processors require special interrupt control logic, create additional modes of system failure in which no interrupts are serviced, and complicate recovery and reconfiguration. A more "democratic" multiprocessor with interrupt masking would permit any number of the on-line processors to respond to the same type of interrupt or alternatively to specialize and designate specific processors for specific interrupt types. Also, in a "democratic" multiprocessor, reconfiguration for interrupt servicing becomes primarily a matter of changing the interrupt masks in the surviving processors.

2.3.8 Error Recovery

The Intermetrics study recommends that recovery from computer system hardware failures be the responsibility of the operating system; i.e., user application programs are not required to incorporate their own recovery procedures. Leaving hardware error recovery as an exercise for the applications programmer is a poor if not impossible use of his specialized skills, but more importantly, it would seriously impair system reliability and the ability to reconfigure the multiprocessor system.

The reasons for this are:

a) Each user would provide his own recovery procedures based on his own notion of what is sensible. Thus the reliability of the system would rest on the weakest attempt at error recovery in the most poorly tested user application program.

b) Uniform response to errors by MPOS could, if proved effective, be revised to obtain a better recovery probability by changing one program. If each user does recovery his own way, then only by changing a significant number of user programs can the system's error recovery performance be improved.

Moreover, system selection of default responses to certain error conditions at system generation may be the best way to obtain uniform recovery procedures. User application programs, nevertheless, must be notified when unrecoverable errors occur so that the user task in control can cancel, degrade gracefully or request the scheduling of an alternate task or job by the human monitor or experimenter involved. The operator must be told of the application consequences of an unrecoverable hardware failure.

2.4 MAN/MACHINE INTERFACE

The Intermetrics study discusses applications of displays as an interactive component in a spaceborne multiprocessor system. There are, however, two more modes of interaction between man and the system than the one the authors describe. The first is used when the man initializes the system following "power on." The second is used when the operator controls the configuration of the system. The third mode is used when the operator finds it necessary or useful to interact with the system.

Many techniques have been used to provide these three modes of operation. The first two modes are usually implemented through provision of an operator's panel and the third through a soft copy display with supporting software. The operator's panel comprises indicator lights and switches. The soft copy display equipment may consist of a cathode ray tube and a keyboard.

For an LSI multiprocessor system, however, the usual techniques for operator's panel construction do not apply. The processor is disparately small compared with physical size and power requirements of indicators and switches. In the RCA-proposed hybrid packaged C-MOS LSI system a CPU/IOU will have only 40 square inches of external surface exposed after mounting--this surface being 1 inch high and 10 inches long on four of the sides. The surface must be shared with the cable connectors used to provide for interunit connection.

A separate operator's panel of the conventional design could be used — another source of problems. Making connections to this panel from the unit is possible, but involves two unfavorable points. Use of a remote panel requires long leads, which add to the capacitance that must be driven by the circuits to which they are connected and whose performance they are being used to monitor. Such added capacitance limits the upper switching rates of the circuits and is contrary to the purpose of using the hybrid packaging approach to reduce capacitance load for high performance circuits. The other point is that with hybrid packaging many logic elements are interconnected within one package and the connections to other hybrid packages are intentionally minimized. Added connections between the logic elements and any control panel would reduce the amount of logic that could be put in a package by using up the connections before using up package space.

The ideal solution for the multiprocessor system would be to provide an interactive console of universal design that connects to the system via the data bus and that can be used as an operator's panel as well as an interactive console.

2.5 STORAGE MANAGEMENT FEATURES

Presently there is general agreement that the memory system will encompass a hierarchy of speeds, capacities and technologies, but no limitation can be put on total size, which would be one factor in judging the appropriateness of system members. The current assumptions are that on-board storage will run to over 4 M words (16 M bytes or 10^9 bits) and that the memory system will support an execution rate of 10^6 operations per second from the processors. Although memory technology is advancing rapidly, the ideal of the monolithic random access store meeting the minimum requirements is not foreseen. Comparisons only can be made in the implementation and management of a large hierarchial and open-ended storage facility.

2.5.1 Evaluation of Proposed Systems

IBM* proposes a system of 15 execution modules (MMUs) each accommodating 16 K 36-bit words (32 data + 4 parity) with access time on the order of 1 μs. Words from any one of these modules are directly available, without buffer or cache, to the addressing unit. Backing the 15 modules are three auxiliary memories with average access time three decimal orders greater and with a total capacity of 3 M words (13 M bytes or 10^8 bits). Thus the ratio of execution to backing store is approximately 1:4.

Still further storage is provided by the bulk store, which consists of peripheral devices feeding in and out of the multiprocessor system data bus. Each bulk storage unit is capable of storing 33 M words (134 M bytes or 10^8 -10^9 bits) on a reel of tape.

Recognizing that program and data requirements have yet to be sized in any detail, IBM has left an open-ended system with blocks to be swapped into the system and data to be buffered out either for data link transmission or in reels of tape for physical transport to the earth. Additionally recognizing that the system may be built and tried out in steps, IBM allows an add-on approach for building the full system.

---

*McNabb, Ref. 15.

Intermetrics* is equally aware of the uncertain sizing of the multiprocessor system, and it proposes module sizes at the M1 level without limitation as to numbers of modules; while at the M2 and M3 levels it gives maximum sizes with no suggestions as to the number of modules Intermetrics might include. Thus Intermetrics is inhibited from detailed discussion of addressing and address translation schemes; nevertheless, it has module hierarchy and management characteristics that can be clearly distinguished from those of IBM.

Intermetrics uses, at its M1 level, a buffer (or cache) associated with each CPU and each IOU. A single M1 module is large, 2 K - 16 K words (10^5 -10^6 bits), and is composed of small blocks of eight words each. It is fast (100 ns cycle time), and it keeps the system busy; every word updated in it must be copied immediately into a corresponding word in M2, the next memory level up, which is one decimal order of magnitude slower and supplies 1 M words (4 M bytes or 10^7 -10^8 bits). After two levels of memory hierarchy, Intermetrics with its large processor buffers is somewhat beyond IBM for capacity and three decimal orders faster. For bulk store, a ROM and R/W combination, Intermetrics proposes a capacity of about twice that of its stated R/W section, which alone was suggested at 134 M words (a billion bytes or 10^9 -10^{10} bits). Such capacity is very generous in view of Intermetrics' citation of an M-D/IBM preliminary estimate of 300 K - 500 K words of execution memory and 7M - 40 M words of bulk storage required for the space station computer system.

Both studies see the implementation of the faster storage units in LSI technology. For its buffers Intermetrics suggests bipolar transistor flip-flops to achieve the 100-ns speeds. For the 1-μs store, the lower-power C-MOS technology is recommended. For auxiliary memory (between bulk and execution) IBM favors bubble memories for their small size, weight and power drain as well as for high reliability. For bulk store IBM recommends high density magnetic tapes, which, although they involve moving mechanisms, offer the option of data link transmission or physical transport.

*Miller, Ref. 16.

2.5.2 Tradeoffs

Given that a memory hierarchy will be established, certain tradeoffs as to the management of the system must be considered. An excellent review of some of these tradeoffs can be found in the Intermetrics study; they are summarized here.*

- Integrated vs. segregated program and data code. Separate busing reduces congestion but increases bus hardware.
- Paged vs. segmented or a mixture of both achieving virtual memory management.
- Alternate storage protection schemes. Paging carries one scheme almost free.
- Interleaving or not. Interleaving can propagate massive failures if one modular participant fails.
- Autonomy for the memory unit v. CPU supervision. The more a memory unit can do its own housekeeping, the more time a CPU has to attend to throughput.
- Alternatives of coding for error detection/correction. The price is in extra logic as well as in extra storage.
- ROM for some program and data storage v. R/W for all. This does not constitute a real tradeoff. Flexibility over long mission life makes R/W preferable.

The list might be extended with two future tradeoffs not listed by Intermetrics although the first was given some consideration in its report:

- Stack storage vs. S/360 dynamic allocation. Although the two methods of referencing stored information are not mutually exclusive, systems orientation will favor one or the other.
- Alternatives in the design of an address translation unit for conversion of virtual address to physical address. Address translation in some form is required for the use of virtual memory.

These tradeoffs are discussed below in varying detail, depending upon their emphasis during this study.

---

*Miller, Ref. 16, pp. 67-69.

It is generally agreed that threat to flexibility militates in favor of all R/W program and data storage and that reliability requirements call for abandoning interleaving in MMU operation. Exploitation of autonomy of all units is a design principle of great potential for a multiprocessor which cannot risk vulnerability of centralized control. This principle is an emergent one, however, and should be studied for the tradeoffs it generates as details of the system evolve.

The ability to establish stacks is attractive to the compiler designer. The structure of higher order programming languages finds convenience in push-down/pop-up lists. For most systems they are created through software, or software with some additional registers for pointer storage. From machines that exist today, it could be concluded that the better compromise is to adapt a machine with conventionally computed addressing (additive components) to establish stacks than to adapt a stack machine to conventional addressing.

A machine which expects to establish some compatibility with S/360 is forced to S/360 address computation methods to implement dynamic allocation of blocks of sequential information. The advantages of stack processing for implementing nested and recursive routines and for compacting memory contents must be realized in software.

Four-bit parity on the 32-bit data word is the generally accepted check on the storage, which must be interpreted as 12.5 percent beyond the sizes given above in order to accommodate this error detection feature. Extreme reliability requirements call for a review of more powerful techniques, i.e., those involving correction as well as detection.

2.5.2.1 Error Correction Coding

Dual-bit error checking and single-bit error correction is presently being employed in the S/370 system for control of some memory errors. However, as

Intermetrics states, this approach still leaves the errors caused by address selection errors such as no-word, wrong-word or multiple-word responses. To eliminate these remaining errors Intermetrics suggests that the only choice may be the multiple copy approach. Neither Intermetrics nor IBM has tried the approach of organizing the memory such that the error control capabilities of the code is matched with the new organization. Such an approach, as discussed in Sec. 3.1.6.2, is capable of economically eliminating the possibility of any single mode failures in a main memory unit causing a system failure.

2.5.2.2 Data Segmentation

Management of virtual memory involves the most fundamental and critical tradeoffs of the memory system. Paging and segmentation separately or in combination are the chief means of achieving such memory, and its effectiveness depends upon selection of various parameter sizes. A good tutorial treatment of the history and operation of paging and segmentation appears in the Intermetrics study*. The gross distinction is that paging implies standard-size blocking of information in a manner invisible to the programmer, whereas segmentation implies blocks defined according to the requirements of the programmer. Thus segmented programs tend to use storage more efficiently than the paged, but the price is in having segments sized with dimensions explicitly stated by the programmer and preserved through compilation. Paging can gain operational efficiency in that it enforces some uniformity of data handling.

2.5.3 Paging and Segmentation Management

Another distinction is found in the fragmentation problem, i.e., development of "waste" space in memory because of storage management policies. Segmentation incurs the risk of not being able to store a segment because the overlay space will not accommodate it. Allocation policies for "checkerboarding," or alternating segments

*Miller, Ref. 16, pp. 85-88.

with unused space, will increase the chances of being able to overlay a segment on demand, but they intentionally induce fragmentation. Such fragmentation by intent is called external fragmentation in contrast to the internal fragmentation that occurs when portions of information in varying sizes are handled in fixed-size pages.

Studies of fragmentation show that quantitative organization of program and data can affect efficiency of storage utilization sometimes very seriously. Both the programmer and the compiler must work with knowledge of memory management policies in order to keep fragmentation down.

Paging and segmentation can be combined in an attempt to achieve advantages of both. Segments are allocated in pages to realize uniform transmissions of data and assurance of placement. Combination schemes will, however, exact some of the overhead of both methods.

2.5.3.1 Critical Parameters

Implementation of paging or segmentation entails further tradeoffs involving certain critical parameters:

- Page/segment sizes
- Memory unit capacity ratios
- Average backing store seek times
- Transport time ratios
- Fetch policies
- Replacement algorithms
- Frequency of page faulting.

A completed system ideally will have these parameters tuned to near optimum conditions. But because of the program dependency of frequency of page faulting there can be considerable variation in the behavior and efficiency of the system under any parameters, and it is realized that best efforts to achieve optimum parameter settings

will result in great inefficiencies under some uses. As the dynamic influence of program interaction has become more apparent, ingenious fetch policies and replacement algorithms have given way to the more straightforward ones. Going beyond the fetch on demand (page fault) and LRU (least recently used) replacement are refinements that can be left for consideration after the operating system and applications programs can be studied for dynamic characteristics.

2.5.3.2 Page Management Illustration

Viewed in greater detail, the effect of a paging scheme on multi-programming can be suggested by an illustrative setup and some postulated parameters. A single CPU can support the simultaneous running of several applications programs. Program A runs until a new page must be brought in. During the transfer a succession of programs can be run with the page fault motivating the commutation between programs (see Fig. 2-6). The parameters being considered are:

- f: fraction page exchanges required by page faulting
- i: number of MMU references per instruction
- N: number of words per page
- n: number of interfault references

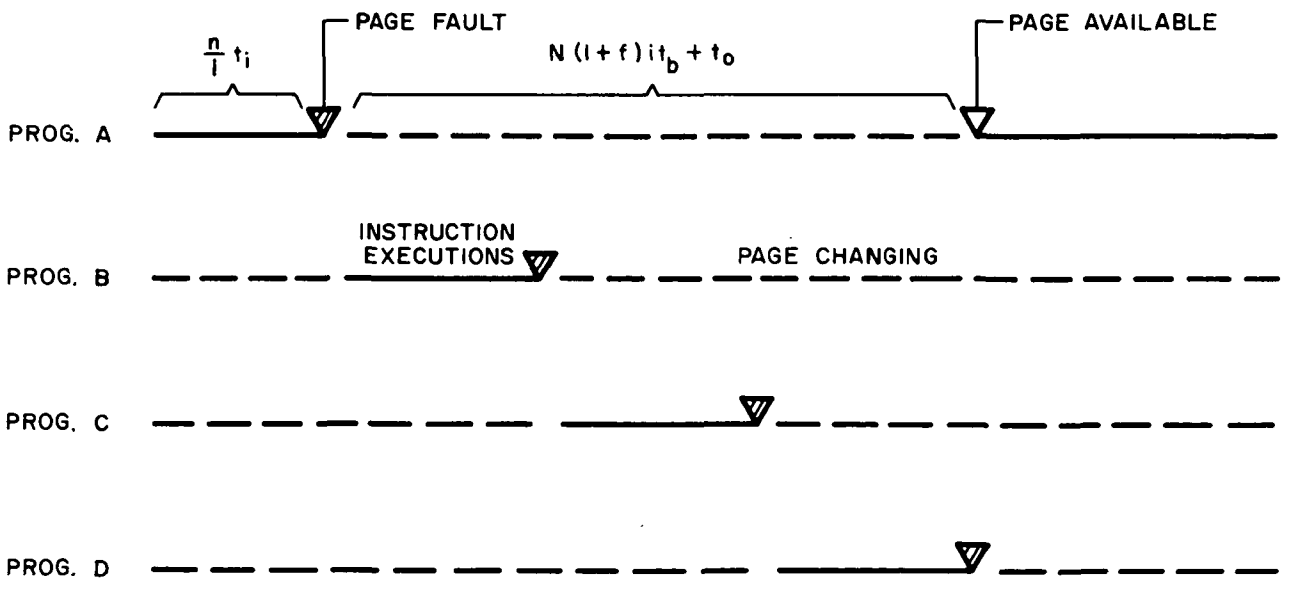


Fig. 2-6. Program commutation by page faulting.

t_b: time for auxiliary memory transfer of word

t_i: average time per instruction

t_o: average time for auxiliary memory setup.

The paging period can be defined in terms of these parameters as:

$$\frac{n}{i} + N\,(1 + f)\,it_b + t_o$$

and the program throughput time per instruction:

$$t_i + \frac{N\,(1 + f)\,it_b + t_o}{n}$$

If the ideal of complete utilization of paging waits is approached, the machine average throughput time per instruction approaches t_i and the throughput rate in Kops approaches $\frac{10^3}{t_i}$. When a simple program is considered alone, its throughput rate, T_p, becomes:

$$T_p = \frac{10^3}{t_i + \frac{N\,(1 + f)\,it_b + t_o}{n}}$$

Neglecting t_o as much smaller than $N\,(1 + f)\,it_b$ assumes the seek, or setup time, is negligible compared to the total page transport time. Under rearrangement, the expression for throughput becomes

$$T_p = \frac{10^3}{t_i}\left(1 - \frac{N\,(1 + f)\,it_b}{nt_i + N\,(1 + f)\,it_b}\right)$$

In this form it becomes clear that theoretical throughput is degraded for the program by a factor that is decreased by number of interfault references (n) if the average time

per instruction (t_i) remains constant. Table 2-1 shows the dependency under a typical set of parameters; the results are plotted in Fig. 2-7. Runs of 100 to 200 instructions without the development of a page fault have been considered as typical for system parameters.

TABLE 2-1. DEPENDENCY OF THROUGHPUT ON PAGE FAULTING

System Parameters Set at	Number of Instructions Between Page Faults	Degradation Factor	Program Throughput (Kops)
$N = 130$	10	.97	6
$f = .3$	30	.92	16
$t_b = 10\ \mu s$	50	.87	26
$t_i = 5\ \mu s$	100	.77	46
$i = 2$	150	.69	62
THEN $N(1 + f)it_b = 338$	200	.63	76
	250	.57	86
	300	.54	92

This illustrative treatment of paging parameters is subject to the criticism that an assumption is made that immediate transfer from one user program to the next constitutes normal operation. Since high priority systems housekeeping routines are usually inserted between applications programs, the figures calculated represent upper bounds rather than the averages. Reality of these figures could be established only from further definition of the system and discovery of program characteristics. Though such definition cannot be supplied at this point, familiarization with the dynamic potentialities can be had through calculations of families of curves resulting from scheduled parameter variations or from simulations.

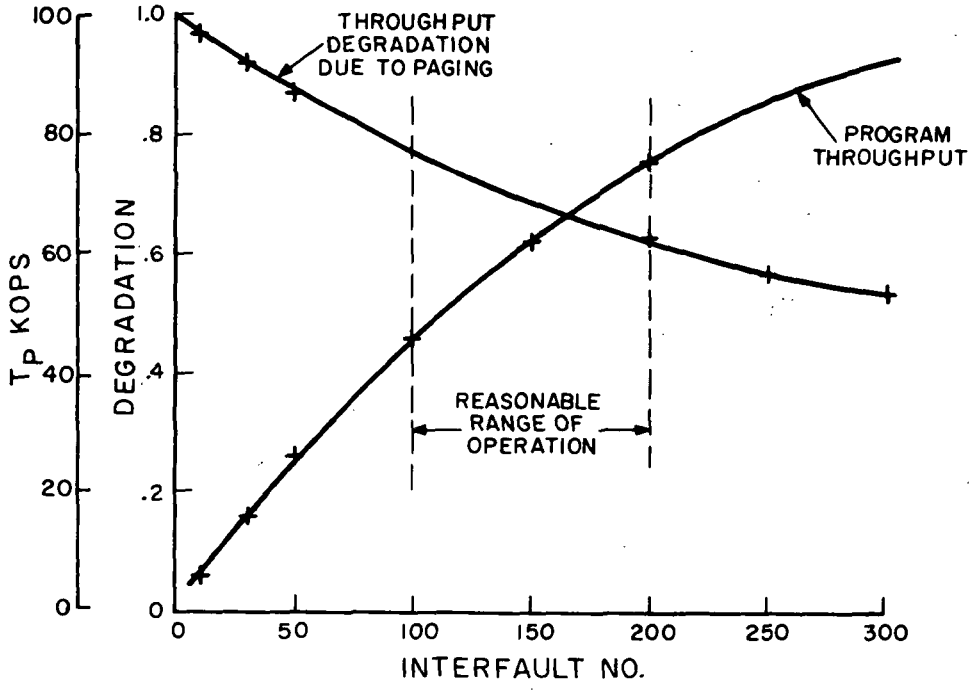


Fig. 2-7. Throughput sensitivity to interfault number.

2.5.3.3 Address Translation

Paging and segmentation are accompanied by the need for generating physical addresses from virtual addresses supplied by the compiler. Mapping of address components is done by an address translation unit (ATU). Components of S/360 type addresses are a 12-bit displacement, which locates a byte within a 4 K byte block, and a 24-bit base which can locate 4 K byte blocks in 15 K different and potentially overlapping areas. A further 24-bit component, used for indexing in some instructions, adds another degree of freedom to relocation. Manipulation of these components results in a virtual address that is satisfactory to user and compiler but must be dynamically adjusted to system requirements imposed on a particular user at a particular time. Final control is maintained in the production of the physical address. Here a

wide variety of protection can be supplied. Two examples are MMU (module) access and access to pages or segments. Denial of use of a module found to be faulty until it is repaired and ready for test must be supplied for long-life, reliable operation; and immediate provision of alternate addressing ensures continued operation on a real-time basis. Equally important is the assurance of program and data integrity. Pages must be associated with certain tasks on a read-only or write-only discipline. They may share pages in common, a convenient means for transferring information between programs, but common access is always under system control. The ATU is the point and means of control.

An ATU, however, has its price. Since it is primarily a function table, it requires its own storage space. It also interposes retrieval and logic steps between CPU/IOU address generation and actual addressing. Content addressed memories (CAMs) can speed the table-lookup but are costly. Tradeoffs can be realized in fitting the CAM size to the management problem. It is not economical to store the entire virtual address translation in a CAM, and the CAM in turn must be managed to make alternative portions available as needed.

IBM has specified an ATU primarily for allocating information in various memory modules -- an important aspect of reconfiguration. Intermetrics has not dealt with the addressing problem in any detail largely because it has left memory capacity open-ended at all levels of the hierarchy and because it has presented an almost equal case for calculated addresses v. list structuring.

Section 3.0

SELECTED MULTIPROCESSOR ARCHITECTURE

This section presents the proposed multiprocessor architecture synthesized from the best concepts from the evaluation of previous studies and extended to show further potential and detail. This section deals with the selected techniques for the multiprocessor and some of the necessary implementations at the system block diagram level.

The salient system features are discussed to emphasize the major concepts embodied in this multiprocessor design. Following this discussion an overall introduction to the system is included to provide a survey of the major units. Brief specification and description of each of the major operating units and their interconnection follow. The hardware and software are presented, and operational concepts are discussed.

3.1 SALIENT SYSTEM FEATURES

3.1.1 Hardware Utilization

A basic premise of this system design is that with the ever decreasing hardware cost per function, additional hardware should be utilized wherever possible to minimize system complexity, minimize software complexity and maximize system throughput. The practicality of large-scale-integration (LSI) coupled with design automation makes possible this decrease. Figure 3-1 illustrates the rapid decrease in cost per gate with advance in calendar year. On top of this trend is the change in breakdown of total costs for an operational system. The total software cost presently is more expensive than the entire hardware cost, and of the hardware cost, the major portion is spent for peripheral equipment. Therefore, even if LSI did not provide this continual reduction of gate costs, the cost of any additional hardware used in the computer mainframe would be swamped by other expenses when the whole system is priced. In recognition of these two facts, RCA has been liberal in its use of additional gates to improve total system throughput.

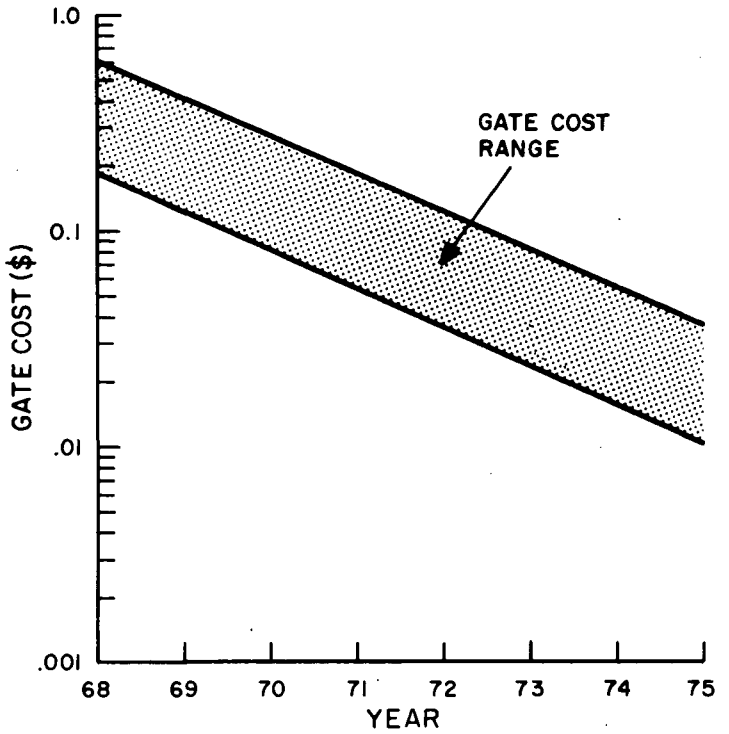


Fig. 3-1. Average cost per gate.

This philosophy can be applied to space hardware only if an ultra-low-power technology like C-MOS is used. From studies made by McDonnell-Douglas as reported by IBM* the cost of hardware for a Space Station Mission is indicated in Fig. 3-2. The hybrid module listed is typical of C-MOS technology. Power is obviously of prime importance.

3.1.2 Modular Design

One of the guidelines during this study was to permit flexibility and growth of each major system module as well as expansion of the overall systems constructed from them. The concept of autonomy of all major units in the multiprocessor system followed. Any system is configured from the three basic module types — central processing unit (CPU), main memory unit (MMU) and input-output unit (IOU). These

*McNabb, Ref. 15.

Incremental Hardware	Assessment	Additional Hardware Cost
1 hybrid package (3000 gates)		
4.18 in^3	\$1,500/ft^3	\$3.63
200-400 mW	\$7,650/W/10 Years	\$1530 → \$3060
0.105 lb	\$250/lb	\$26.25
	Total	\$1560 → \$3090

Fig. 3-2. Hardware cost for space station missions.

modules perform their function independently and cooperatively. The tenet employed is one of "functional split" of responsibilities, a concept which pervades this system design. The MMU performs storage, retrieval, and error checking on data and instructions. The IOU performs all input-output operations for the computer system and is capable of fetching and executing executive code related to all I/O operations and I/O housekeeping. With this capability, the IOU can be completely independent of the CPU. The CPU fetches and executes instructions, interprets code, and performs operations for user and executive routines relating to non-I/O operations and housekeeping.

Thus, the CPU is dedicated to serving the computing sections of user and executive programs, and the IOU is dedicated to executing the input-output and memory data swapping user and executive programs. For any system application, the mix of CPUs and IOUs can be adjusted to maximize total system throughput. An additional benefit of this functional split is that in an error condition an IOU can execute executive code and user code related to computing and thus provide an extra degree of "fail soft" capability.

A further benefit derives from the fact that the IOU design and implementation are almost identical to those of a CPU and consequently maximizes replacement and backup module benefits and minimizes design costs.

The interconnection of modules for a system has been simplified over that found in other multiprocessor designs. There is only one interface. The CPU-MMU interface and IOU-MMU interface are identical. There is no communication between CPU and IOU except through an MMU. This arrangement results in minimal design and complexity of the multiprocessor system.

Programs to be run on this multiprocessor are modularized by the executive. A complete program as presented by the user or executive is a job which is then broken into tasks which are functionally related sequences of instructions with defined resource requirements. Tasks are the smallest defined work package. A task is scheduled, initiated, executed and terminated as a separate entity.

3.1.3 Modular Configurations

Modularity fosters flexibility and reliability when used to facilitate reconfiguration. The general requirement is that there be an MMU for data and instruction storage, an IOU for input-output operation and a CPU for normal computing functions. Under exceptional conditions and because of broad capabilities of the IOU, participation of the CPU may not be necessary. Such operation will not be as efficient as with full processor complement, but it can serve in emergencies.

The minimum operational system that can be formed from the multisystem modules is shown in Fig. 3-3. A simplex system would significantly increase system throughput. This system would have one CPU, one IOU and one or more MMUs, depending on the storage requirements. Such a configuration is shown in Fig. 3-4. The next major step in capability is provided by the CVT multiprocessor configuration with two CPUs, two IOUs, and two or more MMUs. Such a system is illustrated in Fig. 3-5. After the step has been made to a multiprocessor configuration, the system is extendable to include more than two CPUs and more than two IOUs. There is no restriction on the mix of CPUs and IOUs in any one multiprocessor system. The actual mix will be determined by the application programs and needs. A limit of eight MMU

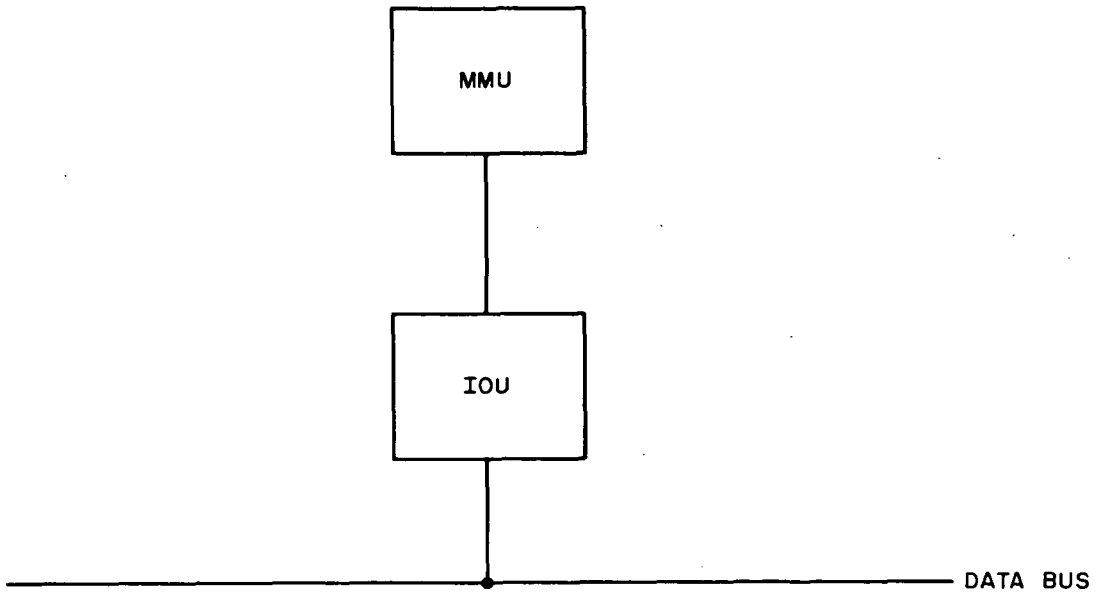


Fig. 3-3. Minimum operational system.

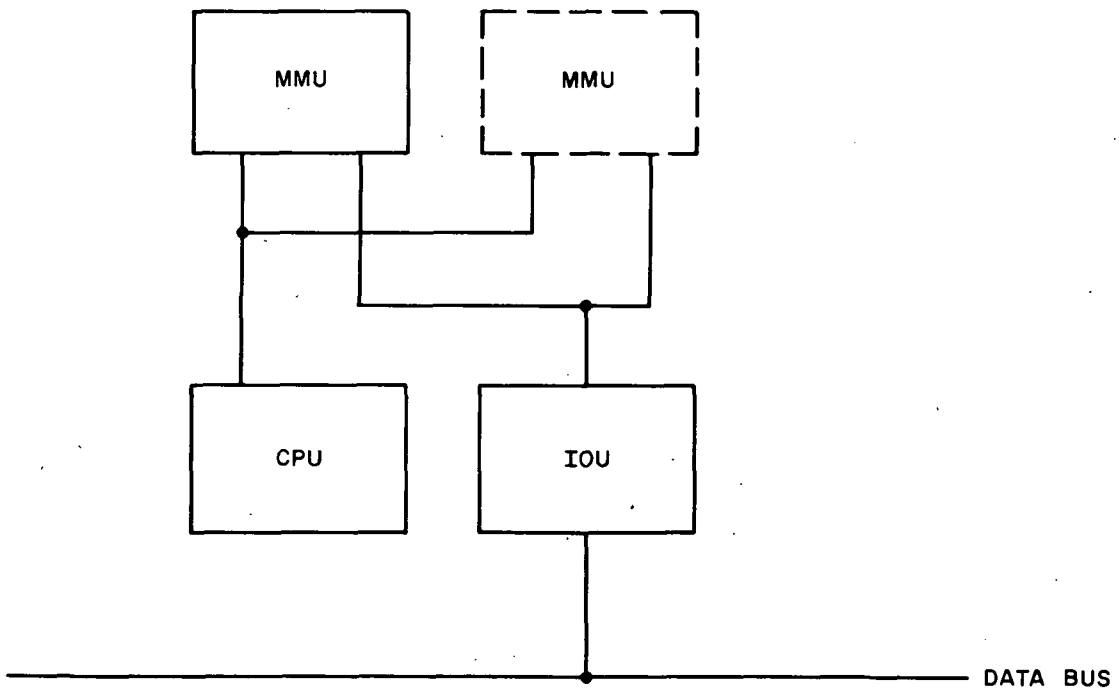


Fig. 3-4. Simplex system.

ports was used in this study to allow independent operation of up to eight CPUs or IOUs in any mix. Though not requisite to the multiprocessor design, a limit of eight appears reasonable. A balanced fourfold multiprocessor system is shown in Fig. 3-6.

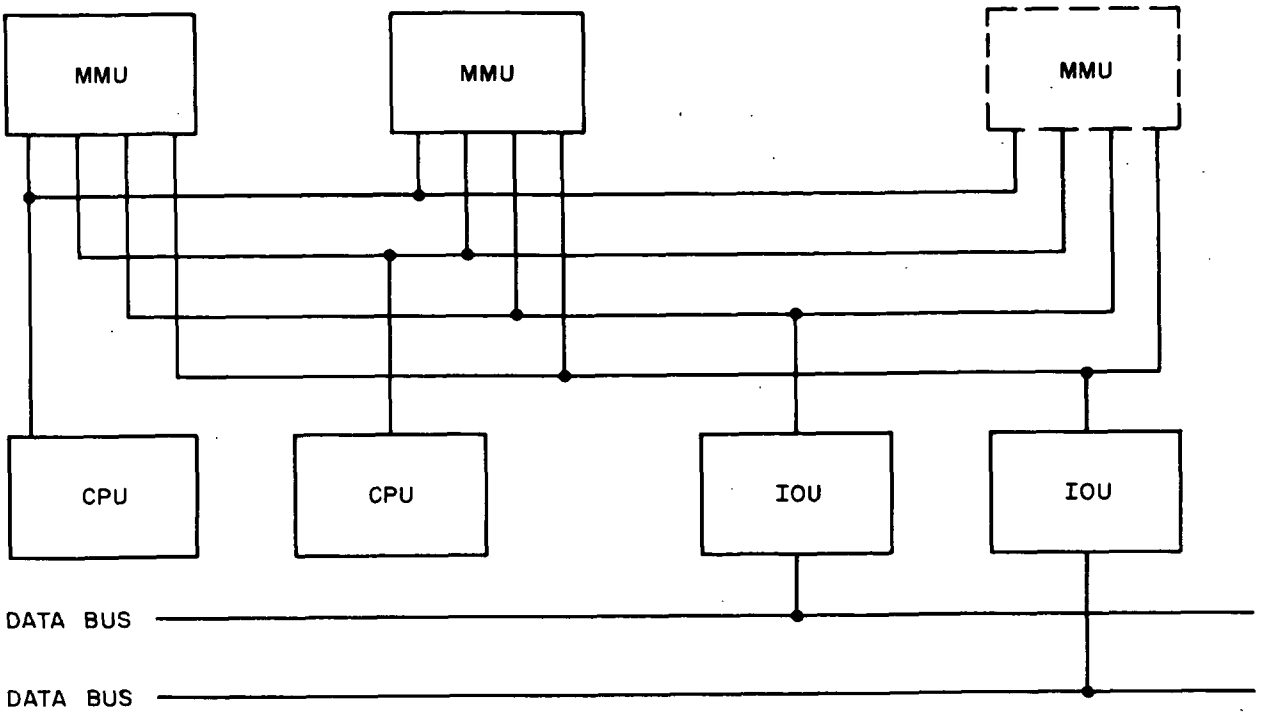


Fig. 3-5. CVT multiprocessor.

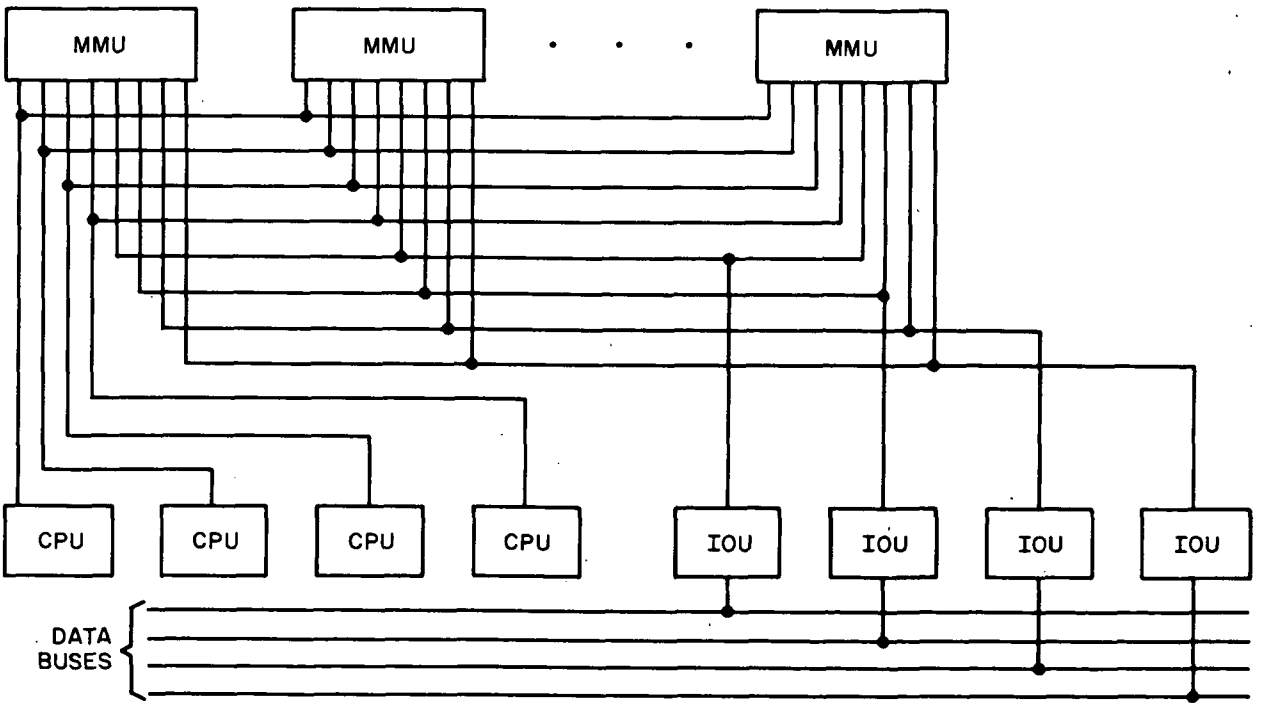


Fig. 3-6. Fourfold multiprocessor system.

In addition to the multiprocessor configurations illustrated above, the basic modules can be used in more conventional multicomputer systems like the master/slave, or federated configurations illustrated in Fig. 3-7.

3.1.4 Virtual Memory Addressing

A virtual memory gives each user the impression that he has a very large memory at his disposal; moreover, it relieves him of memory management (overlay and swapping) problems. The problems remain, but they are taken over by automatic procedures within the system. The user, then, does not see the real, or physical, address but uses a virtual address, which is the result of conventional S/360 base and indexing arithmetic on a displacement. This virtual address must be translated by the address translation unit (ATU) into a physical address by which a memory unit is actually addressed. Synthesis of the virtual address incorporates user and compiler address control, whereas address translation incorporates system control.

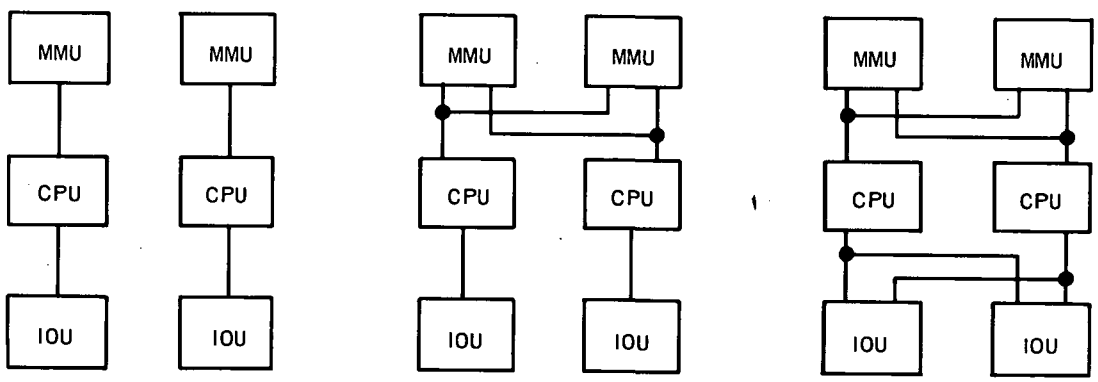


Fig. 3-7. Multicomputer configurations.

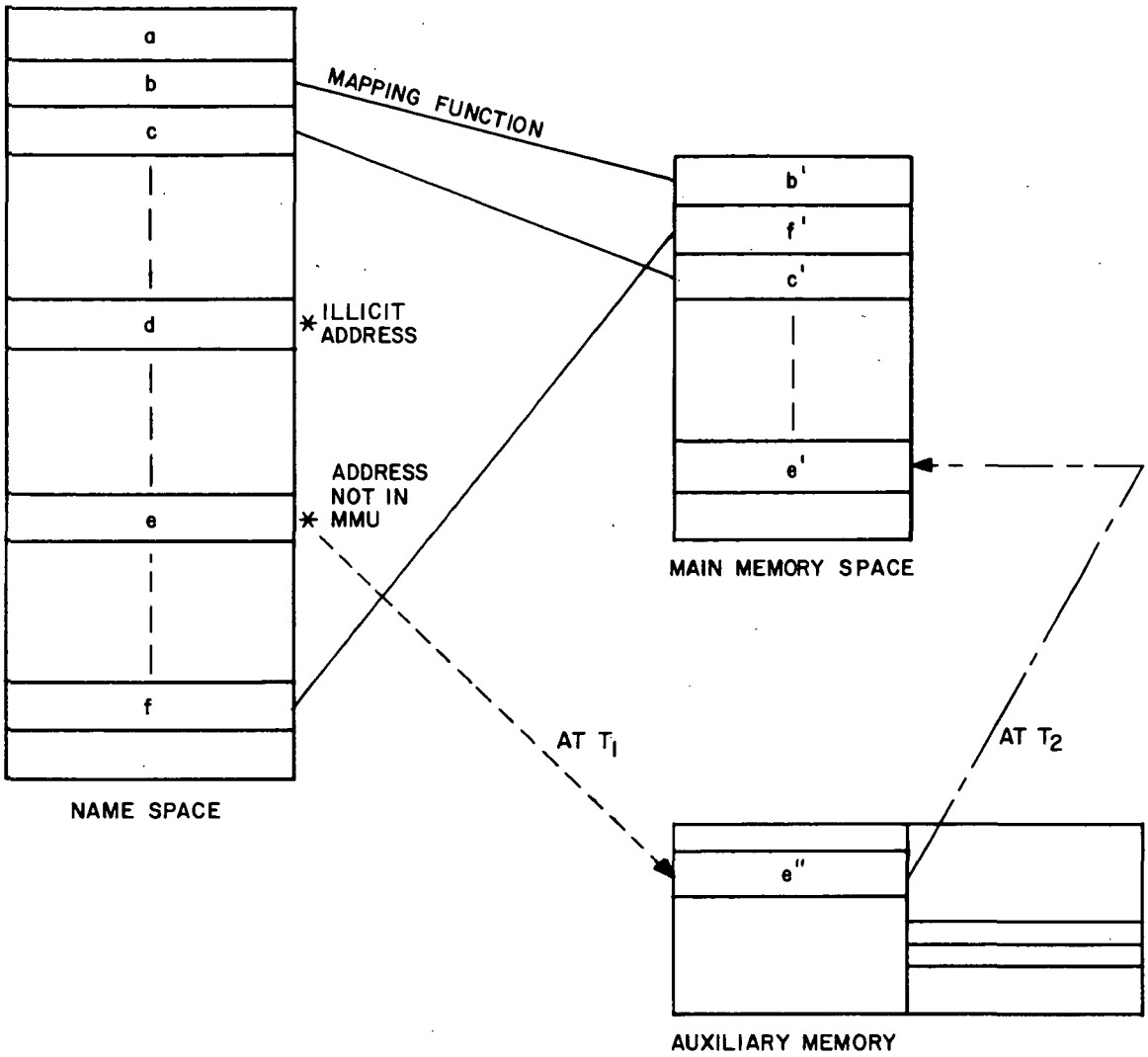
The whole collection of virtual addresses, called name space, is an abstraction. Not all of the names (virtual addresses) can correspond uniquely to memory locations, although any one of them can. Physical addresses by contrast refer one-to-one to real locations in MMUs or auxiliary memories. The mapping function diagrammed in Fig. 3-8 is specified by a map table stored in the ATU. As shown in this figure, not all names are used, and not all can be connected to physical addresses in either of the memories. The first situation is inconsequential, but the second is known as a fault, which can be cleared only if the desired reference can be found in auxiliary memory, moved to main memory, and the appropriate revision made in the translation table.

The data moved from auxiliary memory to main memory involves at least an overlay but possibly a replacement, i.e., the copying out of the contents of a main memory location into "free" space in the auxiliary memory before writing the desired data into main memory. To realize more efficient use of time in transfer, blocks of information rather than individual words are moved when faults are encountered. Pages are uniformly sized blocks of words, and they fit between standard memory boundaries called page frames.

3.1.4.1 Paging and Page Protection

The multiprocessor uses demand paging; i.e., it moves pages into main memory, if it can, whenever a fault is encountered. Whether it first copies out the overlaid page depends upon whether any changes were made to that page during its residency in main memory. Which page frame in main memory is selected for overlay or replacement depends upon when the contents of the page was last used. The least recently used (LRU) discipline is one of the simplest. Although more sophisticated methods have been developed for page fetch and replacement they are sensitive to program dynamics. Initially the simplicity of fetch on demand and LRU page replacement is recommended, but with parametric provision for dynamic tuning.

When the software becomes more definitely specified, it may be shown that segmentation and prepaging of program material are advantageous. Because the task is a fundamental unit of processing and also a unit bounded by the intent of the programmer,



NAME WORD (VIRTUAL ADDRESS)	TRANSLATION	
	AT T_1	AT T_2
a	(UNUSED)	(UNUSED)
b	b'	b'
c	c'	c'
d	(ILLICIT)	(ILLICIT)
e	(e''--NOT IN MMU)	e'
f	f'	f'

ADDRESS MAPPING TABLE

Fig. 3-8. Address translation.

a segment should be associated with a task. Emphasizing this connection, task storage will refer to a segment which is associated with one or more tasks. Pages, then, can be organized to constitute task storage units; and, through software manipulation, these pages can be moved as groups within the memory system.

The benefits of system control of memory allocation, deallocation, and mode and privileges of access via an ATU are considerable. Use of an entire memory unit, or portions of it, can be refused or granted by translation. Such capability makes the ATU key to fast reconfiguration which may be required for routine load adjustment as well as for response to emergencies. Control of access privilege, in addition, is doubly advantageous.

First, areas of memories can be protected not only from intrusion from another task, but also from specific kinds of intrusion. Specification of protection can go beyond read-protect and write-protect to more complex read-only-after-write, write-only-after-read, or timed access, i.e., allowing access only after certain waits. Such protection is more than memory protection; it can be used to guard from software errors like too frequent sampling of data. Further software error protection is attained by extra bits in the data word.

Second, areas of memory can be shared by ATU permission. Sharing of data from a common area saves copying. When read-after-write disciplines are combined with sharing, intertask communication and updating are easily implemented. The multiprocessor system will avail itself of the flexibilities of system control through ATU and will not limit them to MMU assignments.

Through the ATU access to memory pages can be shared or denied. The implication is that the executive must be supplied with, or must develop, information concerning page use during running; and resulting directives must be coded into entries in the Virtual Memory Allocation Lists to be loaded into local ATU on demand. Permissions and protections at the page level are linked to task ID numbers. Word level control is supplied within the data word itself.

3.1.4.2 Address Translation Using a Content Addressed Memory

Theoretically, virtual memory appears as a linear array of 17 M (2^{24}) word locations. In actuality this capacity is more than ample for a single user who, even with files, will accept restriction to a few percent of this figure without being limited. The system will see virtual word spaces in groups of 128, which will fit up to 4 K (2^{12}) page frames in up to 16 main memory modules plus up to 8 K (2^{13}) page frames in auxiliary memory.

Since the processing units (CPUs/IOUs) can use operands and instructions only from MMUs, a directory of where a particular page is at any time must be maintained in the executive areas in the MMUs in order to determine whether it is necessary to have an IOU fetch a requested page from auxiliary memory. Maintenance of the Virtual Memory Allocation Lists is a constant housekeeping activity of the executive which must also notify involved ATUs of changes of status in their stored tables.

Each ATU associated with its processing unit stores the small portion of the page directly that is pertinent to its current activity. The heart of the ATU is a content-addressed memory (CAM) which uses virtual address and task ID information as a descriptor to retrieve the physical address. Formats for input and output address words are shown in Fig. 3-9. The physical address word format indicates the intended manners of relocations of data in the memory system. Removals to different page frames, memory modules, or portions of auxiliary memory can be made by changing the contents of specific fields.

Since the ATU CAM contains only 20 table entries out of a theoretically possible 4 K, there is a table management problem similar to that of a memory hierarchy. At any time the ATU CAM should hold its 20 most-likely-to-be-used translations. Although this criterion can be met for only a small fraction of processing time, an immediate CAM translation represents such a gain in overall efficiency that inefficiencies in CAM use can be tolerated. Tentatively, half of the CAM has been dedicated to executive translations, instantly addressable. The other half of the CAM is assigned to current user tasks so that any running task has several pages of instantly addressable words.

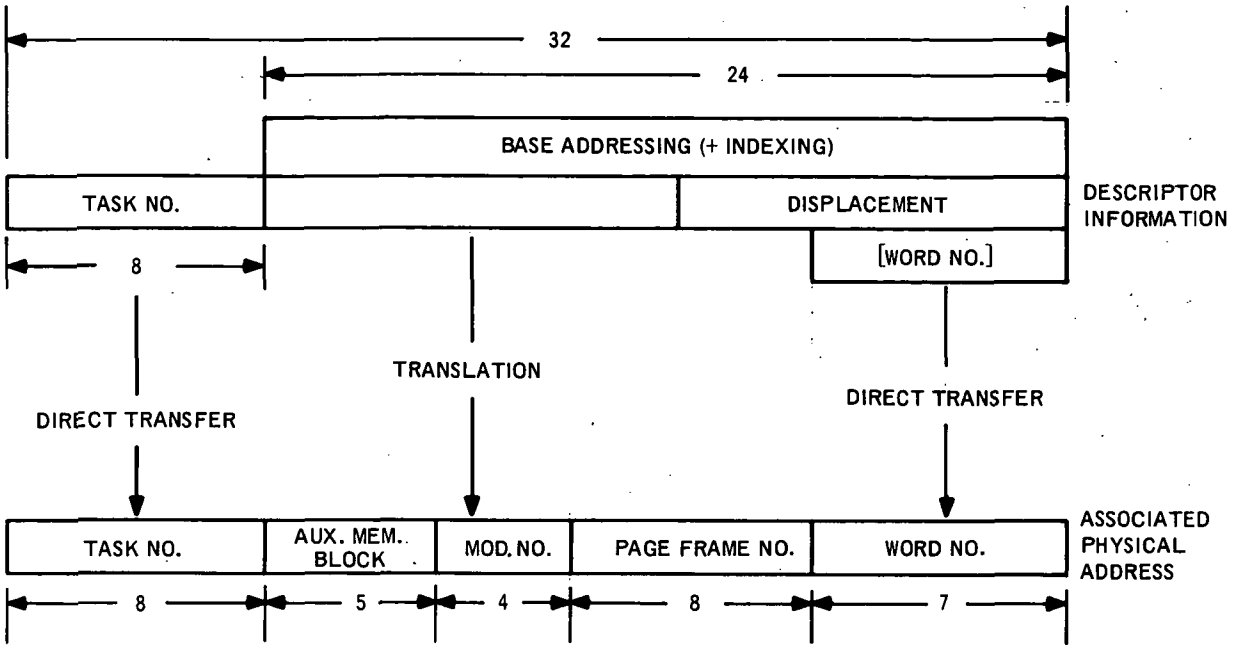


Fig. 3-9. ATU CAM word formats.

Attempt at translation of a virtual address can have one of several outcomes, as shown in Fig. 3-10. If translation is immediately successful, the proper MMU will be addressed with the least delay. The greatest delay will be in page swapping from auxiliary memory to MMU, which involves engaging an I/O processor. While the swap takes place, the user must wait, but the processor can occupy the delay with other tasks. An intermediate situation occurs when the page referenced is in an MMU but the corresponding address translation is not in the CAM. The user task must then be suspended until the CAM can get the reference from the Virtual Memory Allocation Lists in the executive area of the MMU.

Management of the ATU CAM, the updating of allocation lists, and the implementing of system control via the ATU are areas for more detailed study. The ATUs deserve careful attention, as they are key points of system control, especially for reconfiguration.

Does the ATU CAM recognize the virtual address?	Y	Search ATU CAM for physical address	Y	Address MMU		
			N	Notify user that access is denied		
	N	Search page directory for reference in MMUs	Y	Move ATU reference from directory to CAM		
			N	Reference in auxiliary memory?	Y	Start I/O for page overlay/ swap
					N	Notify user and executive of no reference

Fig. 3-10. Physical address search.

3.1.5 Hardware Task Switching

Inherent in this multiprocessor hardware design is the concept of multitasking; i.e., the hardware switches tasks within the system. All tasks are initiated one at a time by the executive scheduler when CPU resources are available. CPUs have several tasks active at any time, although only one task is in execution by one CPU at any time. As soon as that running task is halted, however, an additional task ready for execution in that CPU is then executed. Hardware minimizes time spent in the switching operation.

In this sytem task switching is performed by the multilevel interrupt structure. If a request for attention is set at a higher priority interrupt level, hardware will automatically switch control to the new level and a new sequence of instructions will be executed, indicated by the program counter of the new interrupt level. Almost no time penalty is incurred for this switching since each priority level has a set of immediately available general register and status words. By associating a separate

task with each interrupt level for each CPU or IOU, the automatic switching to the highest waiting priority program ensures that the CPU is never idle.

In a uniprocessor system all events that would cause a priority interrupt are reacted to by the single processor. Any reaction is always by that processor. With multiple processors in a democratic network, several possible situations occur where the event will signal only one processor, and that processor is designated the one to react. In other situations an event will signal all processors with any one, but only one, processor required to react.

The structure of the proposed multiprocessor multilevel priority interrupt control system has been derived from an analysis of situations arising from situations needing attention. According to the analysis, parametric description of interrupt situations, lead to complete definition of hardware/firmware, hardware/control system requirements. Further valuable insight has been gained concerning the interaction of this control system with the executive operating system. In particular, an interlock instruction more powerful than IBM's Test and Set instruction must be used. This analysis has also provided some information on how many priority interrupt levels might be needed, how many similar situations should be handled at each level, and what situations are to be handled at which level.

More details of this analysis are supplied in the following sections, beginning with the description of the interrupt situations that could occur. Section 3.1.5.2 describes the function and coding of the Lock-and-Go (LGO) instruction.

3.1.5.1 Task Interaction

An interrupt causes a change in the sequence of execution of instructions. Interrupt arises either synchronously or asynchronously from external or internal

sources. The change sequence could happen immediately, or it could be delayed until some more convenient time which could occur after any instruction execution is completed and before the next execution is begun.

A Data Exception Branch is also a change in sequence of the execution of instructions but results from operations such as fixed point arithmetic overflow, floating point exponent overflow, floating point exponent underflow, divide error, and significance error. Which exceptions as well as what action is to be taken depend on the task being executed; therefore, each task, when performed by a processor, should have individual control over the recognition of and the reaction of these data exceptions.

An interrupt that occurs anywhere in the system and calls for action by any processor is a global interrupt. An interrupt that occurs anywhere in the system and calls for action by one specific processor is a local interrupt. A local interrupt will usually be designated for a specific processor when it arises from some action of the hardware associated with that processor.

When two or more interrupts occur, the system must choose which one is to be honored first. The choice is made on the basis of established priority - the interrupt with higher priority being honored first. If an interrupt has been responded to but the response has not been completed when an interrupt of higher priority occurs, the system may or may not respond immediately to the later one. If the previous action is suspended in order that the processor respond to the later interrupt, then the later one has the higher priority, which will determine the action to be performed first.

Interrupt is signalled to the processor by a pulse which will set a flip-flop, called a flag. The pulse should endure only long enough for the processor to recognize it but not so long that the response is completed and it looks to the processor like a pulse signalling a subsequent interrupt. The system will await the next interrupt before the cycle described is begun again. The length of time taken by processor response must be less than the shortest time between interrupts or some will be lost.

The processor may wish to delay or ignore certain events because of the times at which they occur. This situation necessitates the inclusion of two types of masking hardware: one, to prevent the setting of the flag, the other to allow the flag to be set but temporarily block the processor's ability to recognize that the flag had been set.

The logic design of interest for this system is shown in Fig. 3-11. In these nets both the flag and the mask can be set or reset by signals sent to the processor from external sources or by signals generated locally.

When it is necessary to activate a task at a higher priority level than the current task, the higher priority task will be readied and the active task suspended. When an active task is suspended, the processor stores the previous contents of the Program Status Word (PSW) registers into scratchpad memory locations assigned to the previously active level for PSW word storage. The new contents to be placed in the PSW registers are read from the scratchpad memory locations used to store the PSW words for the level becoming active. The formats used for the PSW words are shown in Fig. 3-12.

Any time a processor is denied a resource (e.g., a routine), it must wait for the resource to be available before it can gain access to that resource. The processor itself does not have to idle, but may go to some other task while the original task waits. The original task, however, should be notified when a denied routine is available so that it can try again. The processor would then be called back to the task to run the now available routine.

Under certain circumstances a processor may be attempting to respond to a global interrupt and attempt to execute a task unrelated to its current task. Upon finding itself locked out of the routine needed in the response to the interrupt, it would reset the request to run the routine. No notice for callback is needed since the request to run the routine is obviously being fulfilled by another processor.

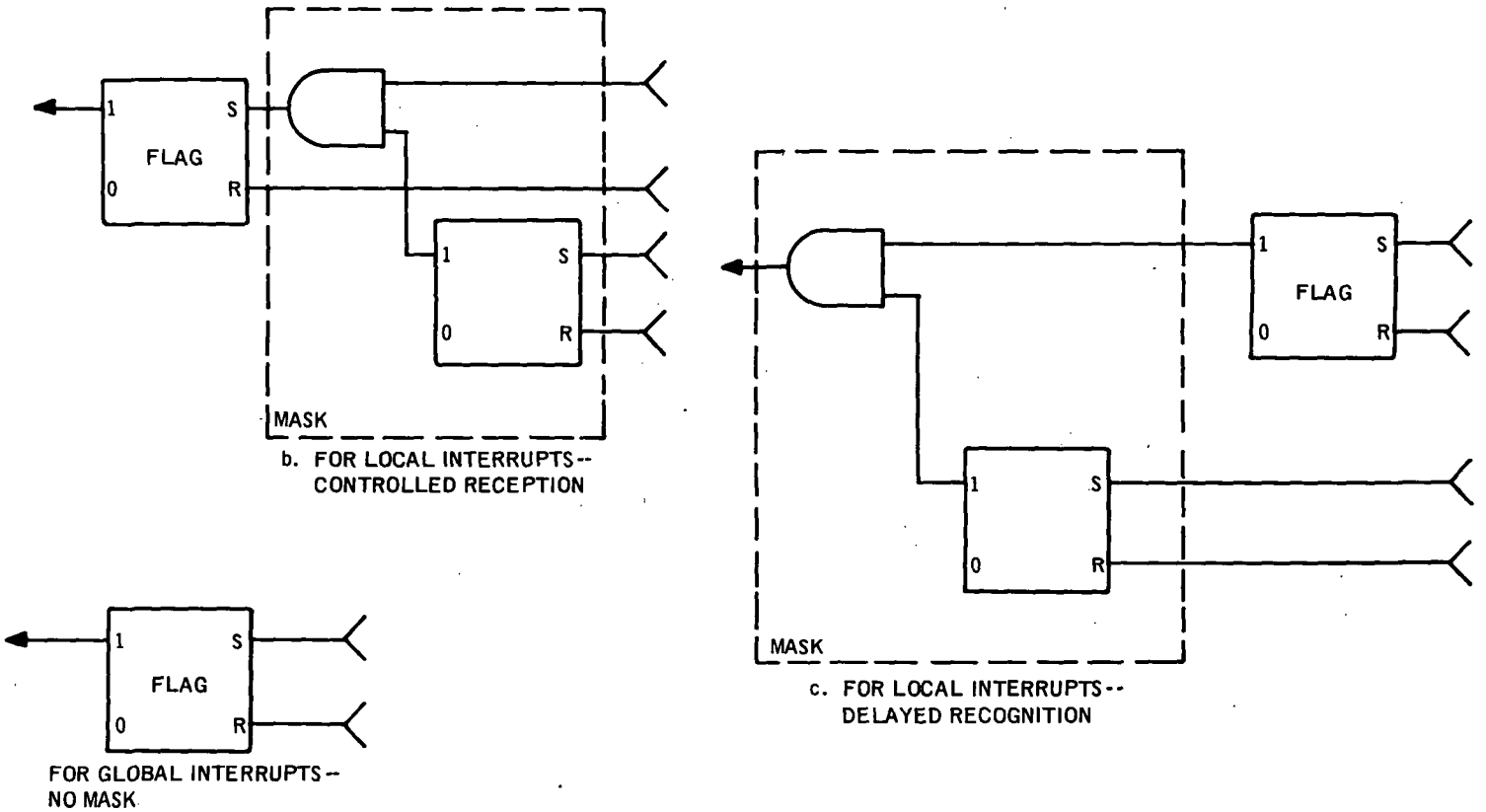


Fig. 3-11. Flag and mask logic nets.

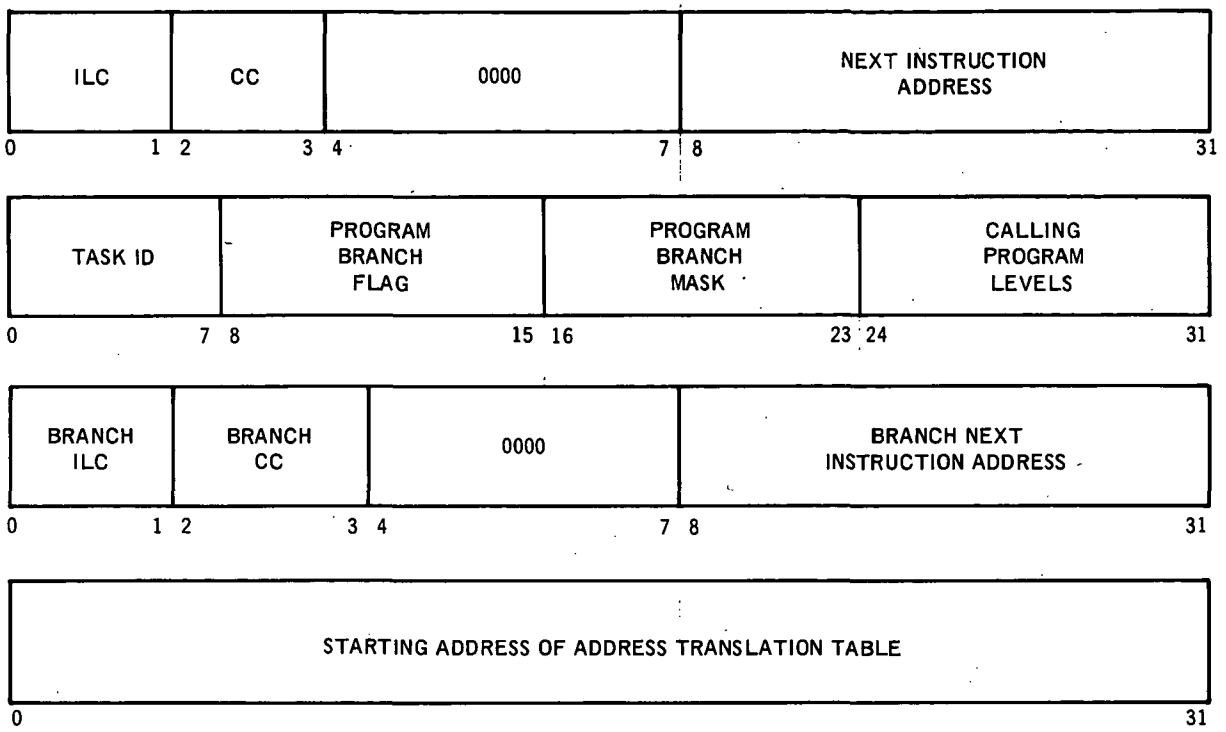


Fig. 3-12. PSW word formats.

3.1.5.2 Lock-and-Go (LGO) Instruction

This instruction provides the processor interlocks needed to avoid conflicts. It enables one processor to lock and unlock a section of code and it will provide information to enable the processor to determine whether it locked the code, unlocked the code, or found it locked. The instruction will enable the processor to calculate, depending on existing conditions, one of three possible entries for the next executable instruction.

The Lock-and-Go instruction always addresses a sequence of protected code which starts within the Locking Word and ends with the Unlocking Return, which resets the program counter to point to the Locking Word. These two words can protect as much code as is needed. Both the address word of the instruction and the Locking Word have fields for task ID. The task ID current in the processor which is executing the LGO is supplied to the MMU by the address. A corresponding field in the Locking

Word is used to indicate the protection status--locked or unlocked, and, if locked, by whom. Other fields of the LGO instruction format contain an immediate address and components of return address.

The sequence of activity of LGO execution can be followed with reference to Fig. 3-13. When a processor addresses the Locking Word, its task ID field is examined. If the field is cleared to zeros, the code is unlocked and the processor may execute the sequence. First the processor locks the sequence by writing its task ID into the task ID field of the Locking Word and stores its return addressing components into a prescribed MMU location. The processor will eventually come to the end of the locked sequence and be referred once more to the Locking Word. This time the task ID field of the Locking Word contains the ID of the current task ID. Comparison of the field with the ID of the address word reveals equality and causes the task ID field in the Locking Word to be cleared and a general callback to all processors (CPUs/IOUs) to be issued. Thus the protected code is unlocked after use. Finally the processor retrieves its return address components, computes its next address, and continues with the task.

If, while the processor was executing the protected code, another processor addressed the Locking Word, comparison of the task ID field in the address with that field of the Locking Word would reject the incursive processor and would turn it aside by causing it to generate another address. The intent is to have it interrupt its current task and work on another until it can have exclusive access to the protected sequence.

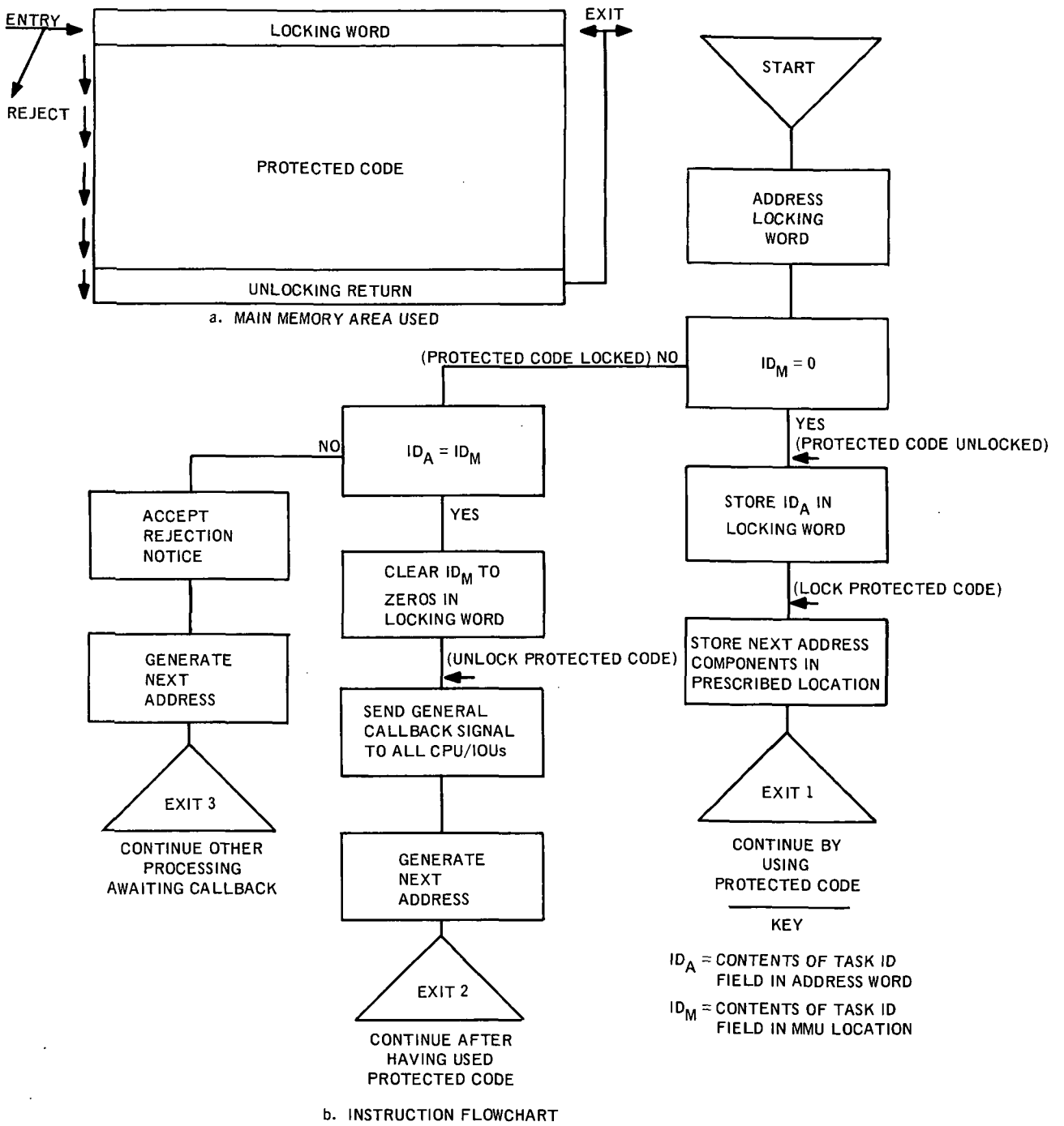


Fig. 3-13. Lock-and-Go instruction.

3.1.6 Reliability

Two forms of redundancy are employed in this system. Multiple similar units (CPU/IOUs) and redundant error control data bits are used with the address and data in the MMUs and interconnections to the CPUs and IOUs. A separate arrangement of error control bits is used with the data sent on the main data bus.

3.1.6.1 Use of Multiple Like Units

More than the minimum required number of CPUs and IOUs have been provided to increase the reliability of the SUMC system. An identifying number is given each unit and used only by the operating system to identify task program assignments to the CPUs.

In many multiprocessing systems the CPU's identifying number is used as a priority code to break ties in the arrival of requests for the same MMU. A particular task under this artificial arrangement is given variable priority as the executing processor may be varied.

To remedy this fortuitous selection each CPU/IOU in the multiprocessor sends, along with other data, the active task ID number which is seen by the MMU as the priority number. Thus priority is linked with task rather than with the processor that happens to be executing it.

3.1.6.2 Error Correcting Code for Memory Units

The error correcting code (ECC) discussed in some detail here is for LSI memory technology. The details for an ECC for plated wire memory are different to protect against different failure modes.

The correcting code chosen was developed at the RCA Laboratories and was chosen for the simplicity and speed with which errors can be corrected and data encoded. The price paid for its high speed, inexpensive operation is twofold.

First, for burst correction errors must be confined to five bits contiguously located in the data word; e.g., a burst of errors is correctable if they appear in the first five bits or in the second five bits of the encoded word and are not necessarily correctable if they appear in the third through the seventh bits. Construction in physical packages of five bits confining all single-mode failures to a specific physical package is a way around this peculiarity.

Second, the error correction capability is limited to one burst of five bits. Design of a physically independent package of five bits ensures that only one five-bit group will fail at one time.

Organizing an MMU from five-bit physical packages and using the suggested ECC will eliminate single-mode failures. Possible failures fall in four classes* and are corrected as described:

1) One or more bits read in error. Since this error by physical construction is confined to a single package, the correcting code will correct the erroneous bits.

2) No response to addressed word. Since each physical package of an MMU is a complete entity, all the other packages will respond and the missing bits will be supplied by the error correction network. With individual power supplies for each package, even a power supply failure will not produce an erroneous output. Furthermore, even a missing module board will not cause an uncorrectable error to occur.

3) Delivery of ORed contents of several word locations. Again with the memory packages, construction will confine any failure of this type to a single package and it can be corrected.

4) Delivery of contents of location not addressed. Since the word delivered would quite likely be a legitimate word, this would be a problem even in a word organized memory with ECC. The planned approach limits any incorrect access to a single byte and corrects it.

*Miller, Ref. 16.

Each memory must be a complete operating unit to eliminate all single mode failures; i.e., each package comprises the full address decoding for a 32-bit address word. It receives five bits of data, will read out five bits of data. Each package receives five incoming data bits and corrects them by using the contents of the encoded word. Output errors are corrected in the same way.

A complete MMU word consists of 32 data bits, 7 status bits and 11 bits for ECC -- making a total memory word length of 50 bits. If each 5-bit memory package is constructed with 8192 words of 5 bits each, as shown in Fig. 3-14, then a total 8192-word MMU will require ten memory packages.

3.1.6.3 Use of the Burst ECC for Correcting Internal Bus Errors

The use of a burst ECC on the internal bus can, as in the main memory, correct all single mode failures. By organizing the bus such that data sent between the units are bit-serial groups over a number of parallel wires, it is possible to select a burst ECC that can correct for all errors that could occur in any one-bit serial group. This technique is similar to that proposed for the main memory units. Moreover, the same grouping of bits can be used for the internal bus system as that used for the main memory units. Identical grouping permits the same design for all ECC logic units. It is possible also that data sent from a CPU could be encoded at the CPU and corrected only where the data is handled. For a store-and-then-fetch operation, correction would occur at the input to the MMU, at the output of the MMU, and upon its return to the CPU.

3.1.6.4 Use of a Burst ECC for Main Data Bus Errors

Three alternate approaches apply to use of ECC for the Main Data Bus. Using the arrangement for the internal bus would result in a multiwire bus. The same effective data transfer rate would require a 13-wire bus, would result in greater weight and size, and would require many more connections.

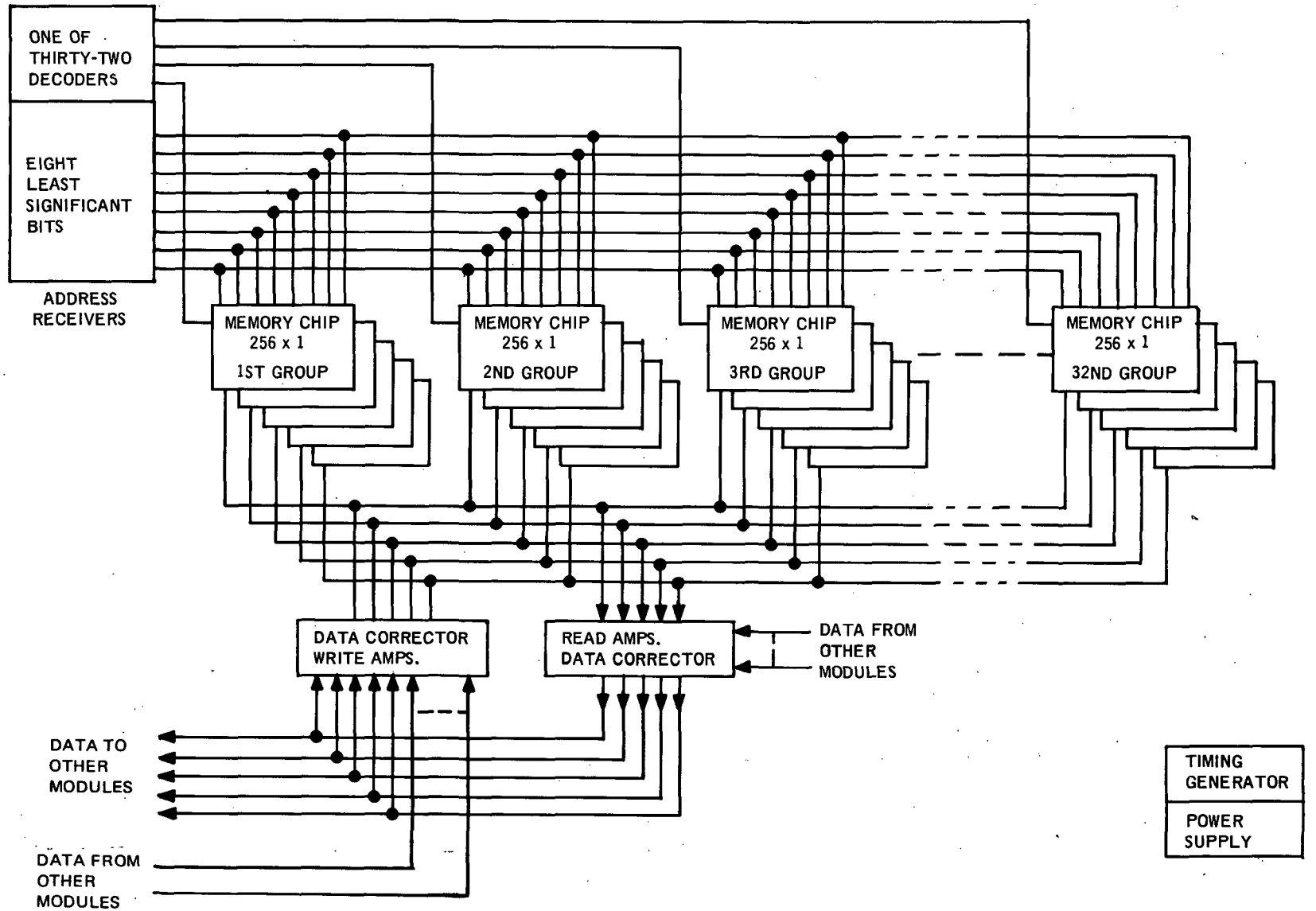


Fig. 3-14. Typical memory package (8192 x 5).

One advantage of the multiwire bus is that error correction systems are more efficient than those applicable to a single wire subsystem. Any burst of errors would usually affect a larger number of bits in a row than any practical error correction code can accommodate.

A more practical approach would be to use an error checking code. An error would initiate a request for a repeat of the message. This approach can be implemented in a number of ways. The two most practical schemes result in a transmission efficiency of 50 percent.

The most efficient encoding scheme for an error checking code is a two-dimensional Hamming code capable of one-bit error correction and two-bit error detection. This scheme, when used with a large data block, approaches the 50 percent transmission efficiency and 100 percent error detection capability. *

The other scheme calls for the receiving unit to return the message to the sending unit. ** The return message is compared bit by bit with the original message for any discrepancy. Repetition of message corrects the error.

3.1.7 IBM S/360 Compatibility

3.1.7.1 Motivation for Supporting S/360 Compatibility

Many valuable programs potentially useful for NASA in SUMC have been, and will be, developed, tested and used in IBM S/360 computers. While the SUMC missions might be so novel that no existing S/360 applications programs will fit their requirements, much general purpose software that could support these missions has been developed by IBM, by IBM users, and by software product firms. Moreover, the savings in development costs and the increased assurance of reliability obtained by using existing, tested software are significant.

*Peterson, Ref. 18.
**Proch, Ref. 19.

A further argument for a significant degree of S/360 compatibility is that much of the program development for SUMC software components (operating system software and application programs) could be done by NASA and NASA contractors using S/360s and thus reducing the number of SUMC computer systems needed to support program development.

Examples of existing, or proposed, IBM S/360 programs whose conversion to MPOS/SUMC could save considerable amounts of money over total redevelopment include:

Programming language compilers (e.g., for PL/1)
Data base management systems (IMSII, GIS)
Linear programming packages (LP 360)
Simulation packages (e.g., GPSS)
Trajectory equation solvers
Guidance and control programs

3.1.7.2 S/360 Compatibility Alternatives

"Compatibility," as used in computing, however, is a term almost entirely devoid of meaning. When computer people use this term they are usually talking about characteristics of programs and their data, which could be more accurately and more meaningfully described otherwise. An operational definition of "convertibility" provides a terminology with meaning and measure for discussing compatibility.

An informal, operational definition of convertibility is: the cost in dollars and linear time of replacing one computer system with another. Thus, we can give the term convertibility meaning by saying that it is a property of a given computer-system-for-computer-system replacement and that the measure of that property is the cost of carrying out this replacement. The measure of convertibility in replacing a given S/360 model 40 with an identically configured S/360 model 50 can be held to zero.

Emulation (hardware/software) is the most successful method of minimizing conversion cost or maximizing convertibility.

Job Control Language is a programming language; therefore Job Control Stream conversion is a special case of program conversion. Assembly Language programs are invariably more costly to convert than FORTRAN, ALGOL or COBOL programs; usually complete reprogramming is required. Table 3-1 shows the range of conversion procedures applicable in conversion between S/360 and SUMC, together with the relative cost of each approach.

TABLE 3-1. CONVERSION COST ESTIMATES

Amount of Rework by User	Impact on Cost of new System's Software	Conversion Level
20-80%	+0%	Algorithm to Algorithm The original problem analysis remains constant but the old source program is discarded, resulting in complete reprogramming. A re-implementation of the algorithm and some modification of the algorithm is likely. Full advantage is taken of new compilers, new operating system, and new hardware features. This is the only conversion procedure applicable to on-line communications applications programs.
1-10%	+20%	Source Program to Source Program The algorithm remains constant and the original source program is recompiled using the software of the replacement system. Ideally no manual conversion is required prior to this recompilation although program controlled conversion is often used. A strong functional similarity between operating systems is required. Some advantage is usually taken of new operating system features and new hardware features simply because the new compiler tries very hard to do so. This approach rarely is applicable to assembly language programs.

TABLE 3-1. CONVERSION COST ESTIMATES (Continued)

Amount of Rework by User	Impact on Cost of new System's Software	Conversion Level
0-1%	+40%	Object Program to Object Program Source program remains constant and old object program is reprocessed by the new system's linkage editor and/or loader. Requires new operating system to have same interface as old as well as identical functions. User subset of the computer architecture must be identical or else simulated.
0%	+5%	Executable Program to Executable Program The loadable (executable) program remains constant. Requires the execution of both the original application program and the original operating system. All differences in computer architectures must be detected and trapped during program execution. Programs with timing or special device dependencies cannot be emulated. No advantage is taken of new operating system or new hardware; Throughput increase as good as can be expected. There are two basic types: (a) Stand-Alone Emulation (Simulation) New computer system is dedicated to emulating the old computer system. (b) Concurrent Emulation (Simulation) New computer system multiprograms the emulation of the old computer system concurrently with execution of programs prepared for the new system.

Concurrent emulation of OS/360 under control of the native MPOS could be a disaster in space. Even if the behavior of MPOS were predictable and its reliability near perfect, the resource allocation and scheduling characteristics of OS/360 could adversely affect both its predictability and reliability. Moreover, any applications run under control of OS/360 would themselves be beyond the "knowing" of MPOS and its monitoring, error detection and error recovery facilities.

The very programs one wishes to run in this mode may require some amount of rework (i.e., conversion) in order to achieve acceptable levels of performance, reliability and resource integrity, and the requirement (or the desirability) of using alternative peripheral devices in the multiprocessor environment may require conversion of S/360 programs to interface with these new devices.

Given the present level of reliability of OS/360 (extended also to OS/370) it is unlikely that the "concurrent emulation" mode would provide an adequate degree of confidence for a space environment. It is presently reported that OS/360 software releases contain approximately 1000 detected, documented errors.

3.1.7.3 Two Levels of S/360 Convertibility Chosen for Support in SUMC

It is reasonable to assume that source program conversion for S/360 programs and data file conversion for S/360 data files can be so automated as to make the conversion cost per program (per data file) small compared with that for original development, and that the ratio of conversion cost to development cost can be kept below 10 percent on the average, given the current thinking on multiprocessor architecture (i.e., the S/360 architecture is a subset of SUMC). The reason these conversion costs exist are:

- Differences in the behavior of MPOS compilers v. OS/360 compilers.
- Differences in the interfaces for object programs between MPOS and OS/360.

The proposed approach represents the best compromise between two extremes: running the original program under the old operating system via emulation of S/360 hardware with no opportunity to improve either this program or the operating system and complete reprogramming to take advantage of the features and behavior of the operating system.

It should be emphasized that this level of compatibility will be worth the effort it takes to minimize the two differences cited. Using this minimization of differences as a design criterion results in much better tradeoff decisions in the design. Designers would have to justify the introduction of novel features which increase the cost of converting S/360 program and data files.

It would be short-sighted to do anything in the design of SUMC which precludes stand-alone emulation of OS/360 at the multiprocessor architecture interface. SUMC systems, electronically isolated from any space station controlling activities, would support this interface. The multiprocessor would employ a single CPU as a uniprocessor and the IOUs as S/360 multiplexor and selector channels.

This S/360 emulation capability would be valuable in validating the S/360 programs prior to their conversion to equivalent programs running under control of MPOS.

3.1.7.4 Effects of Storage Allocation Techniques on S/360 Compatibility

The designers of Multics virtual memory and storage allocation system make a good case for a two-level addressing scheme in which pages are contained in segments, and segments correspond to natural subdivisions

of an application into program modules and data files. The additional complexity and an additional table access during address translation are justified by the following advantages:

- Use of logically distinct segments eliminates the need for users to manage program overlays, given a sufficiently large segment space.
- Simultaneous sharing of information (data or program modules) between tasks is readily effected if that information is managed as a segment.
- Moving data between address spaces is not a prerequisite for passing the data to another task; a segment pointer can be passed instead.

The multiprocessor design has managed to retain the first two capabilities without requiring a two-level addressing mechanism. Imposing a useful segmentation mechanism on SUMC would negate much of the carefully attained S/360-SUMC convertibility. This feature was felt to be too great an asset to sacrifice for a memory management technique that may quickly become technological obsolete. It should be noted that in multiprocessor terminology "task" is similar to Multics' "process" and "Program Structure List" closely resembles the Multics' "directory structure."

The third advantage cited by Multics (moving data between address spaces) is questionable. If the data are in a large table or a file (as in Multics) then the cost of the move could be prohibitive. However, in the multiprocessor system, files are managed by a conventional means, for convertibility reasons. Therefore the penalty for moving data in order to make it addressable has been accepted in order to gain the advantages of IBM S/360 and simplified address space management.

3.2 OVERALL SYSTEM INTRODUCTION

The final system selected for the multisystem architecture embodies all of the characteristics and features discussed in Sec. 3.1. The block diagram appears in Fig. 3-15. The resulting system can be characterized by four major features: 1) high-throughput democratic multiprocessor, 2) fault tolerant operation, 3) compatible with third generation IBM 360 type computer systems, and 4) open-ended.

The high-throughput democratic multiprocessor is achieved by use of several techniques. The design employs virtual memory addressing to present adequate addressing space to the applications programmer and eliminate any need for him to plan overlays. Moveover, by using paging to achieve the virtual memory addressing, a severe memory allocation problem in a multiprogramming environment is solved. With the fixed size pages, bringing active program pages into main memory is a low overhead function. To ensure maximum utilization of the hardware modules, hardware multitasking is employed. Each CPU has several tasks ready for execution at any one instant. As one task is halted in the CPU, its hardware rapidly switches to a new task and proceeds with execution of that task. When this new task is halted another or perhaps the first, again begins execution. In this manner the CPU is switched among several tasks in minimum time. This system in general will maximize total system throughput at the expense of individual task throughput. However, the executive scheduler using information concerning task deadlines may after task execution to follow minimum/maximum time-line schedule critical to a mission. Section 3.4.3.2 discusses task schedule management in detail. This technique is compatible with the system being task driven as opposed to being CPU driven.

Fault tolerant operation is likewise achieved by several techniques. All single-point communications are eliminated. The failure of any one module type does not affect the other modules except to notify them of the failure. Related is the fact that each communication path has only one "boss." Each CPU and IOU has its own bus to

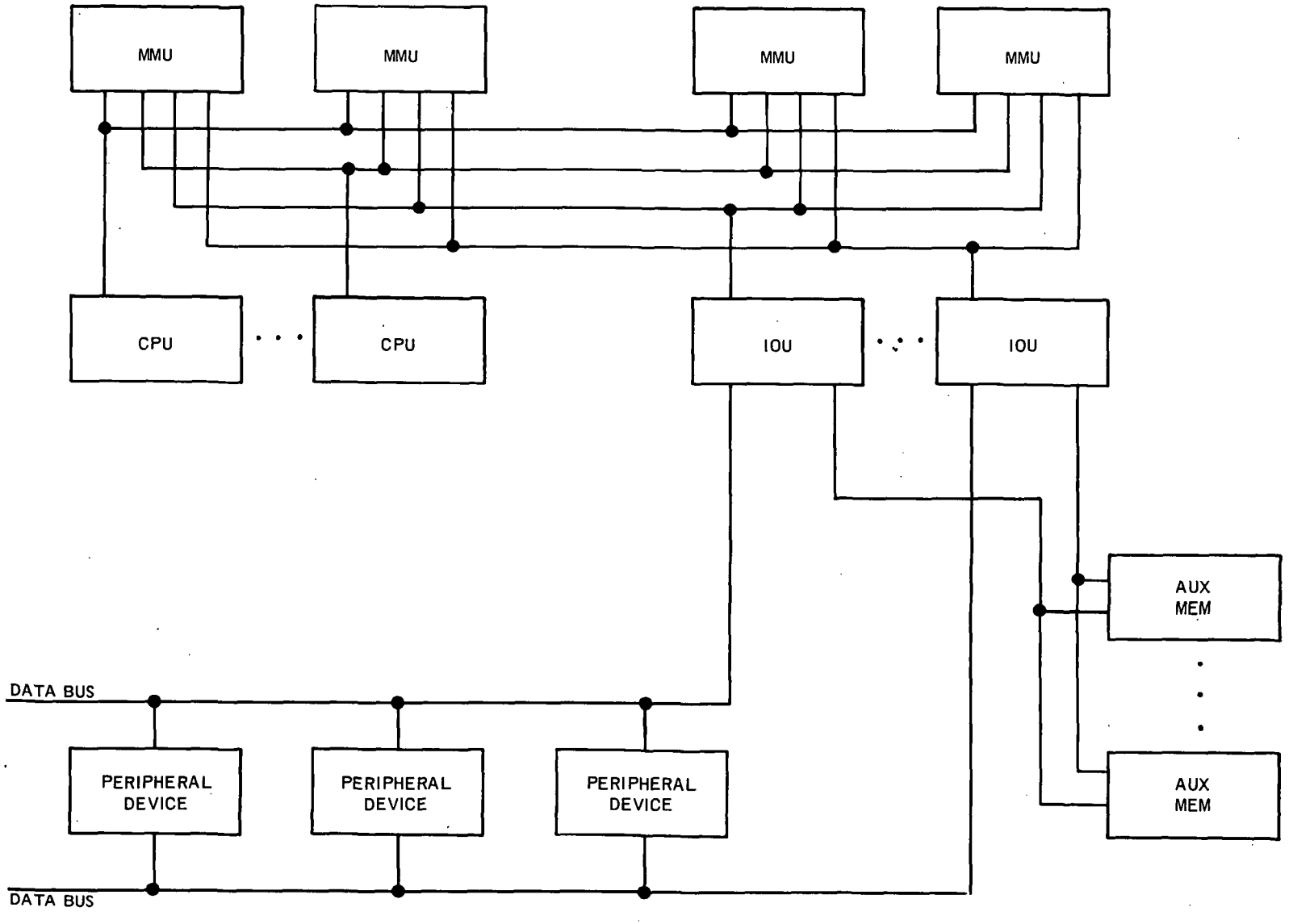


Fig. 3-15. Multiprocessor block diagram.

the MMUs and each IOU is in control of its own data bus to the outside world; furthermore, each unit type is independent of similar units. The only interaction is the vying for executive space at aperiodic intervals. Units communicate with different type units (CPU with IOU) through the MMU in which communication areas are set up so that communication is ensured with minimum overhead. Because MMUs are central to the entire system, communication error correcting coding is used to monitor and correct their input and output. A single error is automatically corrected whether it is in address information, data, or status, and the system is notified. Error checking is also built into the MMUs at the word level to ensure legitimate usage of each word. Bits stored in the word indicate type and legal use of information stored there (data, instruction, nonexecutable, etc.). For each MMU access, these indicators • are checked by hardware and any error reported to the originating module (CPU or IOU).

Compatibility with IBM S/360 computer systems is obtained by adopting the IBM 360/370 addressing and instruction format. IBM 360/370 becomes a subset of the multiprocessor defined here. In the CPUs additional hardware has been added to facilitate use of IBM 360/370 software. Provision has been made for emulation of IBM 360/370 software. In this mode the multiprocessor becomes a uniprocessor, with the CPU emulating the IBM 360/370 CPU and the IOU performing the channel command programs only.

Open endedness is obtained from modularity and "functional split" of responsibility discussed in Sec. 3.1.2. Additional redundancy can be added later either by additional units or within existing units. System throughput can also be increased with the use of additional units in the system and by capitalizing on advances in technology. Any unit type, because of its independence, can be implemented in several technologies without adversely affecting the total system. Taken to the extreme, each unit in the multisystem can be implemented in a different technology, and the system will still function.

3.2.1 Physical Building Blocks

The multisystem is configured from four major units. This section will briefly summarize the characteristics of each unit type. Section 3.3 gives detailed information.

The MMU must provide storage for both programs and data. Each MMU is accessible by both CPUs and IOUs via independent paths. In case of simultaneous access, the MMU determines priority based on a decode of the task ID number.

The CPU must execute user and executive code for the "compute" type of functions. It sequences, fetches, and executes instructions for non-I/O types of instructions. The CPU handles all instruction and data formats of IBM 360/370. Each CPU contains 16 levels of interrupt, with each level having a separate set of general registers and status words. Of the 16 independent program levels, seven are for independent user tasks and nine are dedicated to entering the executive. This scheme allows each CPU to handle quickly the executive needs of the user tasks. Normally one CPU does not need to communicate with another. Each CPU translates its own virtual addresses to physical addresses. In the event of a translation fault, the CPU reverts control to an executive interrupt level designed to handle this fault.

The IOU must execute executive code for I/O related functions, which include the paging and memory management functions required. The IOU must also translate its addresses as does a CPU. The IOUs like the CPUs, are independent of other IOUs. Any IOU can perform I/O tasks for any CPU. The IOU must also provide all system communication with the outside world. It must take control of the main data bus to perform the I/O functions with the peripheral equipment. A separate, conventional digital interface is provided by each IOU to interface with an auxiliary memory.

The auxiliary memory adds storage capacity for user and executive code that may be required by the tasks currently active in the multisystem. As tasks require address space not currently in the MMUs, the IOUs locate the requested code and retrieve it from the auxiliary memory. Bubble or holographic technologies are strong candidates for the implementation of the auxiliary memory.

3.2.2 Operating System

The executive system for the multisystem computer coordinates functions required to initiate user programs, to supply support functions required during the execution, and to schedule resources needed by these programs. The key feature of this multiprocessor operating system (MPOS) is a division of a total operating system into CPU-related and IOU-related functions. The CPUs then independently execute the compute-type functions and the IOUs execute the I/O-type functions as needed. Parallelism tends to minimize resource conflicts.

3.2.3 Interconnections

The multiprocessor system modules communicate over one type of interface. The CPUs and IOUs have identical interfacing to the MMUs. By employing a standard interface, maximum interchangeability in the event of failure is ensured. A CPU can be switched to a memory port previously used by an IOU with no modification needed. Heavy reliance on the memory interconnection dictates that an error correcting code will be used on the internal bus to implement correction of any single failures. The code will be used for correction at the CPU-to-bus, IOU-to-bus, and MMU-to-bus interfaces.

3.3 MAJOR OPERATING UNITS

3.3.1 Main Memory Units (MMUs)

The MMUs provide intermediate storage between the small, fast scratchpads in the processing units (IOU/CPUs) and the bulk storage of the auxiliary memories, and they are the instruction and operand storage for the processing units. They are also the chief communication links between processors and so hold messages, status reports, and executive tables for immediate reference by any processor. Because of their vital role in system and individual processor operation, each of the MMUs is more than a large set of registers with necessary addressing logic, and includes error checking/correcting and interrupt signal generation hardware to make versatile responses to actions of the processing units.

Gross specifications for the MMUs are shown in Tables 3-2 and 3-3. They are subject to refinement as the rest of the system is more fully specified, and its demands more accurately known.

TABLE 3-2. MAIN MEMORY SYSTEM GROSS SPECIFICATIONS

Number of units	8-16
Capacity	65 K-262 K words (3.2 M-13 M bits)
Physical realization	Plated wire/C-MOS
Features	Word oriented Priority service Protected areas Message areas Preferential areas Dynamic storage allocation Error checking/correcting Interchangeable ports Interrupt generation

TABLE 3-3. MAIN MEMORY UNIT GROSS SPECIFICATIONS

For Each Unit	
Access time	.6-.8 μs
Cycle time	1 μs
Physical size	425 in^3
Weight	7 lb
Power required	10-15 W
Storage capacity	16 K/8 K ($2^{14}/2^{13}$) words 256/128/64 pages
Page frame capacity	128/64 words
Word size	50 bits

A block diagram showing major functional units of an MMU is shown in Fig. 3-16.

Although the processing units (CPU/IOU) will retrieve and store words individually from the MMUs, chief movement of contents is in pages and is effected as a type of I/O operation between MMUs and auxiliary memory via the auxiliary data bus. Figure 3-17 shows the relative rates of transfer in the paging operation. Memory ports interface in a standard manner with all processing units so that any IOU can change pages on demand of any CPU.

Service of access request is on a first-come-first-served basis except when the MMU is busy or the accesses are simultaneous. Then the MMU may queue requests and serve them according to priorities based on task ID. Previous studies have shown that queueing for common memory access need be no serious problem in a multimodule memory system; therefore, schemes for buffering, interrupting, and notifying the system of a queue await further knowledge of specific timing and dynamic use of the multiprocessor system.

Control and checking information is stored in each word. Word format and an explanation of the control and status indicator bits are shown in Fig. 3-18. These indicator bits are maintained and checked by the CPU/IOU served.

Certain program logic errors can be detected in the MMU access mechanism of the CPU/IOU by means of these indicators. Since software errors are more frequent, and often more difficult to diagnose, than hardware errors, memory word status indicators can improve SUMC reliability through increased ability to detect software errors and through improved capability in isolation of software faults.

Any MMU can generate interrupts arising from

- Unit failure -- e.g., power
- Unrecoverable data transmission error
- Program exceptions -- e.g., illegal use or specification of address or data bounds
- (Overuse -- e.g., queue building).

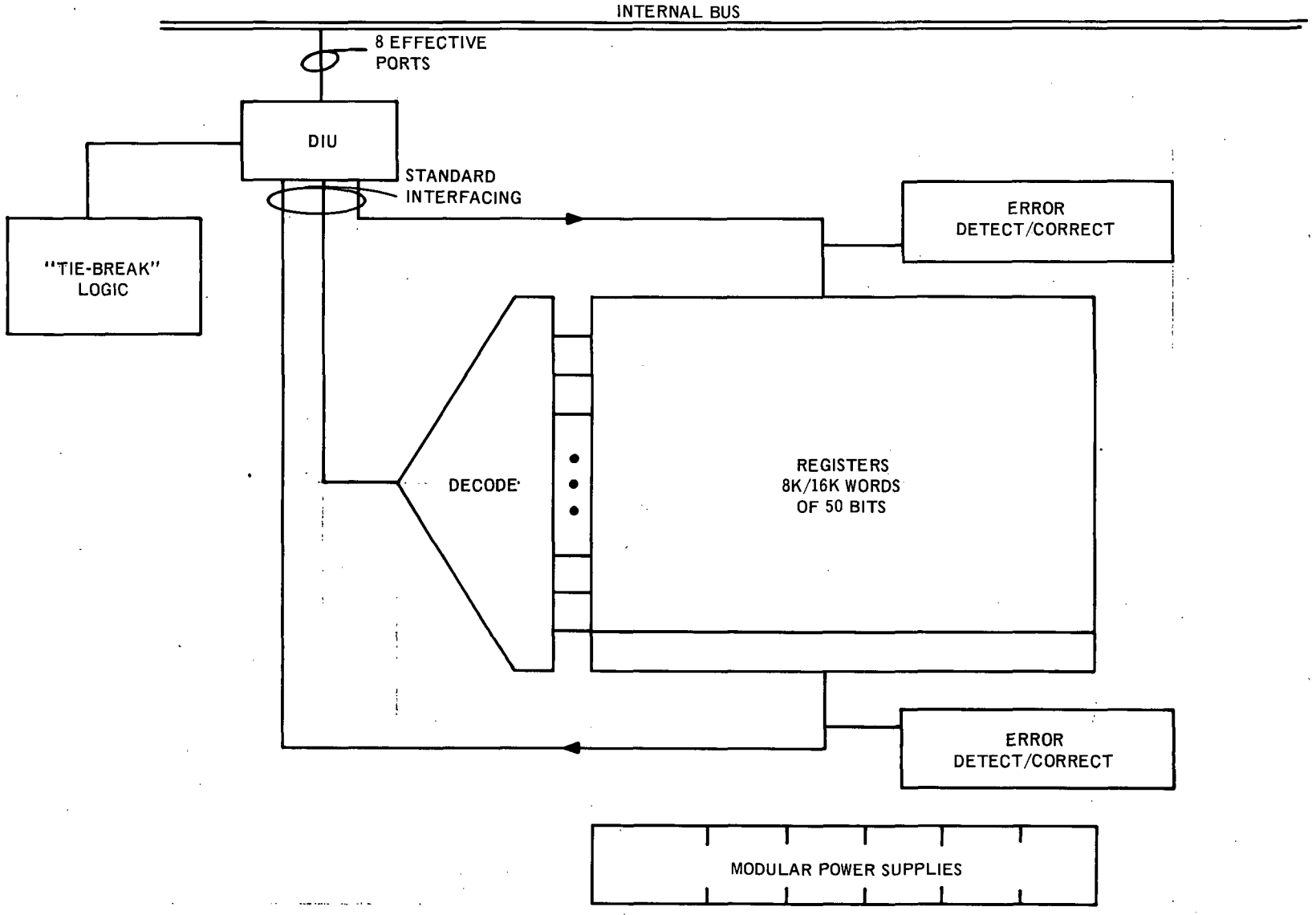


Fig. 3-16. Main memory unit block diagram.

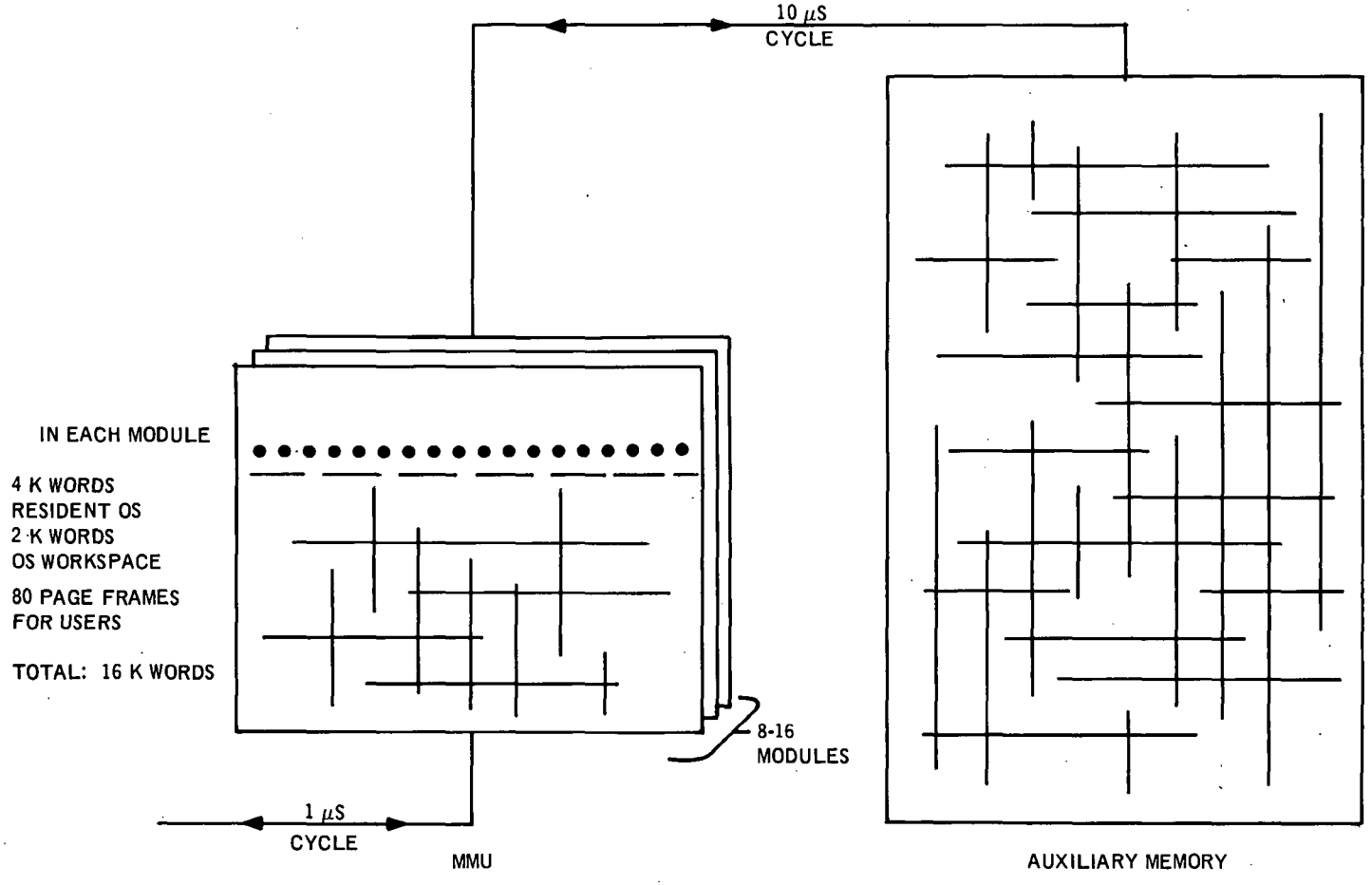


Fig. 3-17. Paged data flow.

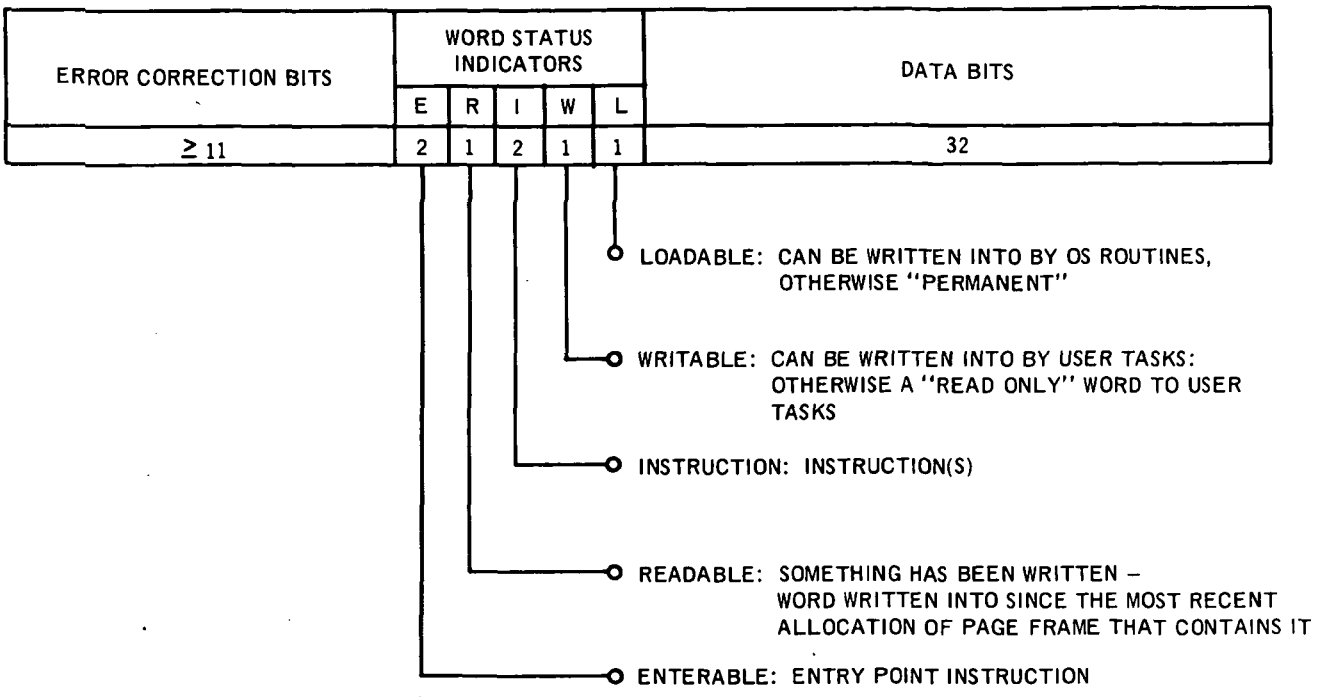


Fig. 3-18. Memory word format.

The first should generate a global interrupt. The last source of interrupt is tentative and both its generation and whether it can be handled locally must be determined by need.

3.3.2 Central Processing Units (CPUs)

The CPUs perform the bulk of the computational work of the system. Although they must support various housekeeping operations, they are relieved of many such burdens because of the autonomy and versatility of the IOUs in combination with the MMUs.

Gross specifications for the CPUs are given in Table 3-4. These specifications, too, are subject to change as the entire system becomes more definite.

TABLE 3-4. CPU GROSS SPECIFICATIONS

Number of Units		1 to 4
Word Size		32 bits
Instruction Complement		Full S/360 set plus system instructions
Add Time (RR)		1 μs
Physical Size		95 in^3
Weight		2.4 lb
Power Required		5 W
Scratchpad		
Cycle Time		200 ns
Register Complement		448
General	256	
PSW	64	
Floating Point	128	
Multiprogramming Facility		
Processing Levels		16
User	7	
Executive	5	
System Engineering	4	
Features	ROM firmware control ATU for virtual addressing S/360 emulation Six processing states Sophisticated interrupt management	

The block diagram of one unit shown in Fig. 3-19 indicates the familiar SUMC processor design with its basic cycle of multiplexer registers, arithmetic unit, and scratchpad with parallel floating point unit. Interface with the internal data bus to the MMUs is via standard interface, allowing interchangeability of units. Interface with the main data bus is via DIU. Connection to the auxiliary memories is direct. Control sequences are ROM controlled.

Three structures shown are less familiar: the Byte Transform Unit, the Interrupt Control, and the Address Translation Unit. The Byte Transform Unit is a byte-wide logic unit with multiplexers for handling SS-type instructions. Typically these instructions determine the course of processing the next byte of a variable-length string according to results of processing the current byte. Byte-oriented instructions are vital to executive operations.

The Address Translation Unit (ATU) accommodates virtual addressing by translating the locally computed address into the system physical address. Chief substructure of the ATU is a content addressed memory (CAM) which holds 20 words, part of each of which is the descriptor by which the word is retrieved. Among other substructures is a logic net which can be set up by system parameters to decide on the propriety of the physical address. Thus the ATU functions partly as a memory-protect unit. Rationale for and function of an ATU have been treated in a previous section (Sec. 3.1.4), and the application of address translation in a dynamic paging situation will be shown later (Sec. 4.3).

The third new structure, Interrupt Control, will contain an interrupt register consisting of 16 flags (one for each level of interrupt) and a mask register to facilitate temporary or permanent suppression of each interrupt. Interrupt levels arranged by priority are listed in Table 3-5.

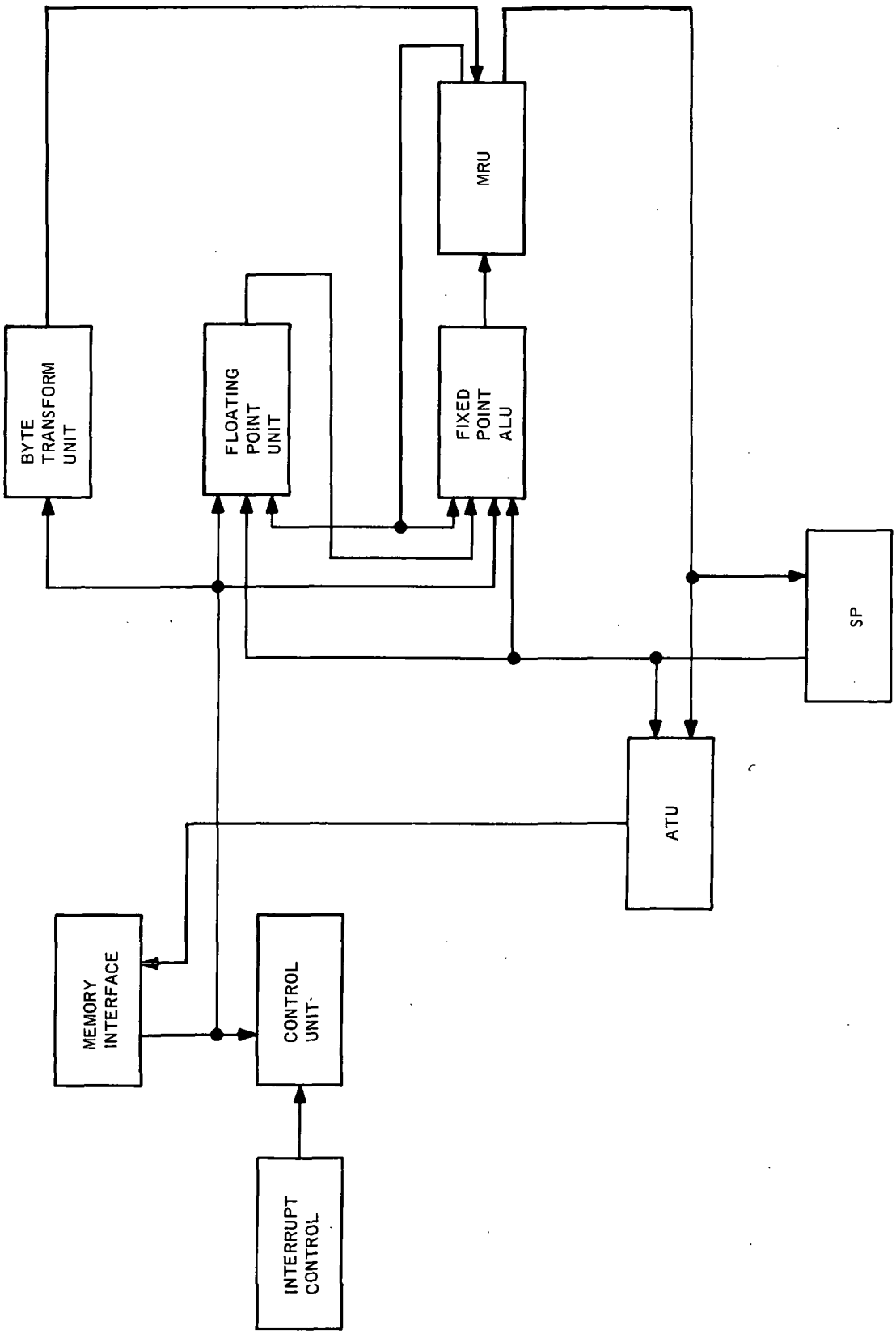


Fig. 3-19. CPU block diagram.

TABLE 3-5. CPU INTERRUPT PRIORITY LEVELS

Routine Function	Priority Level
Power Failure	15
Module Failure Entry	14
Mode Switching	13
MMU Code Check Failure	12
Executive Entry	11
Task Control	10
I/O Request	9
Page Fetch	8
Interval Timer	7
User Level 6	6
User Level 5	5
User Level 4	4
User Level 3	3
User Level 2	2
User Level 1	1
User Level 0	0

Priority

An interrupt of higher priority will, when honored, cause the Active task to be Preempted to the Ready state from which it will be Pended to wait to complete itself at least until after the higher priority interrupt has been serviced. It will be necessary to Terminate it if, when Pended, it still claims needed resources. If no other interrupt higher than the previously current level has come in during the intervening servicing, then a return by interrupt to the previously current level can be made. If it has been Terminated, then all resources needed must be available so that it can become Active again. At all times a CPU will attempt to process at the highest task priority level for which there is an outstanding interrupt.

3.3.3 Input-Output Units (IOUs)

The IOUs serve the system as chief movers of bulk information. They load MMUs from peripheral devices and auxiliary store for processing, and they move the processed information out. They forward interunit messages from CPUs to the operator. Moreover,

they are autonomous; i.e., they do their own housekeeping as well as operate independently on I/O tasks.

The I/O task is the unit of I/O processing, and the page is the unit of data transfer. Although only one word at a time can be moved by a single IOU, the IOU will, despite possible interruption, strive to complete a page transfer just as a CPU strives to complete a task.

An IOU executes subchannel programs upon orders left in "mailboxes" by CPUs. To do so it must when unemployed automatically and regularly check for orders. It will also poll the devices it is servicing according to a schedule maintained by the Periodic Calling Routine (see Sec. 4.2). Results of transfers must be recorded in the data structures of the executive; consequently, the IOU spends a significant portion of its processing capability in maintaining system records described later (Sec. 3.4.2). Residual processing is entailed in maintaining the IOU as a "good citizen" of the multiprocessing system. It performs its own executive functions, chiefly allocation and deallocation of resources and participating in orderly bus use. Under reconfiguration it is responsive to system demands, even to performing as a limited CPU.

The block diagram of Fig. 3-20 shows the portions of CPU hardware and capability used for the IOU. Building the IOU identical to the CPU in LSI hardware is economical in that uniformity of design and construction outweighs loss in unused CPU features. There is, moreover, an important benefit in having full CPU hardware present and wired into an IOU, and this benefit lies in the ability to incorporate an IOU readily into the system as a CPU.

The memory interface will be standard with that of the CPUs and so allow interchanging units, a flexibility needed for reconfiguration. Memory addressing techniques are uniform throughout the systems; hence the ATU will be a standard adjunct. The scratchpad requirements for both IOU and CPU are the same for 16-level interrupts.

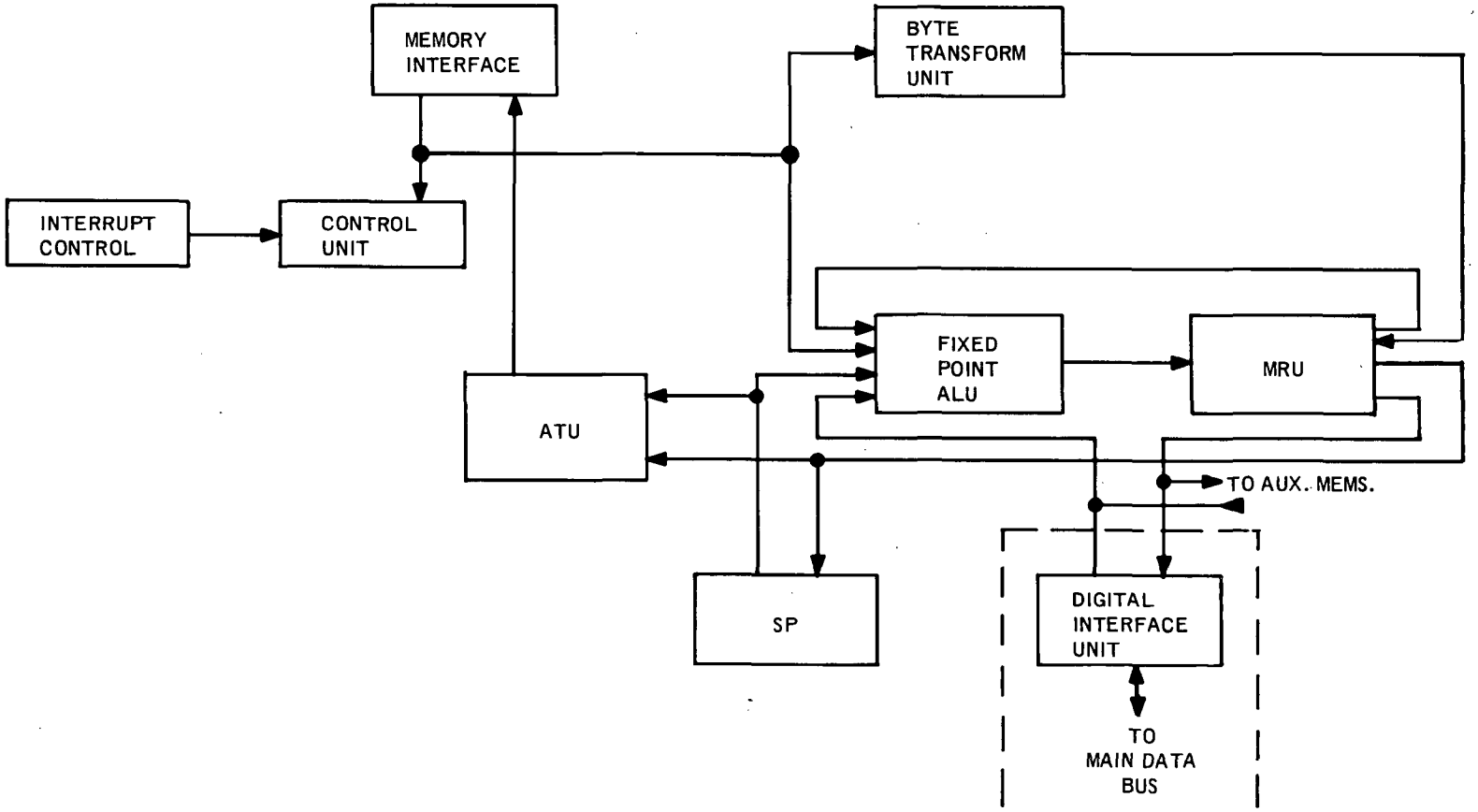


Fig. 3-20. IOU block diagram.

What differentiates an IOU from a CPU is the firmware, or contents of the control ROM. Although the ROMs of both types of unit will have regions of identical code, which directs micro-operation for the common S/360 type instructions, they will differ as to specialized instructions. The IOU, furthermore, is committed to its own interrupt structure and even the top four interrupts can have different implications for I/O.

Gross specifications for the IOUs are shown in Table 3-6. Again, these are subject to refinement as the rest of the system becomes more fully specialized.

TABLE 3-6. IOU GROSS SPECIFICATIONS

Number of Units	1-4	
Word Size	32 bits	
Instruction Complement	S-360 type Less floating point Plus system instructions	
Throughput		
Add Time (RR)	1 μs	
Word Transfer Rate	Up to 100 K words	
Physical Size	425 in^3	
Weight	7 lb	
Power Required	15 W	
Channels		
Auxiliary Memory	1, 2	
Subchannels	1 to 7	
Scratchpad		
Cycle Time	200 ns	
Register Complement	448	
General	256	
PSW	64	
Floating Point	128	
Features	ROM firmware control ATU for virtual addressing Sixteen interrupt levels Alternate standard S/360 I/O interface	

The IOU accepts and responds to interrupts as does the CPU. Its tasks, although of different function, have the same four processing states and transitions initiated by interrupt. The 16 interrupt levels ordered by priority are given in Table 3-7.

TABLE 3-7. IOU INTERRUPT PRIORITY LEVELS

Routine Entry	Priority Level
Power Failure	15
Module Failure	14
Mode Switching (Reconfiguration)	13
MMU Code Check Failure	12
Executive	11
Page Program	10
Task Control	9
Page Fault	8
7 Subchannel Programs	1-7
Subchannel Servicing	0

Priority

Seven task levels, similar to user levels, are employed for setting up and terminating use of individual subchannels. The lowest level is used for routine servicing of the seven subchannels as described in detail in Sec. 4.2. Discipline on the main data bus, over which each IOU must poll and exchange data, is maintained locally in each IOU by the Periodic Calling Routine and globally by use of the Lock-and-Go instruction, which allows unique access to a key instruction without which the IOU cannot use the main data bus.

The IOUs are connected to the auxiliary memory units via a separate data bus over which information in pages can be brought. The separate bus precludes interference between page exchanges and traffic with the peripheral devices.

Internally the IOU is connected to the MMUs via a memory interface, the standardization of which makes interchangeability of units possible.

3.3.4 Auxiliary Memory

The sizing of auxiliary memory units must follow further specification of the storage needs of the system. Throughout this study auxiliary memory has been regarded as an "expansion joint" which has provided adequate buffering between bulk storage and the MMUs, as well as providing an adequate number of page frames for virtual memory management. Conservatively its capacity will be between one-half and twice that of the combined capacity of the MMUs -- 32 K - 524 K words. Its transfer rate must be about 100 K words/s. Auxiliary memory is envisioned in modular units which can be switched on line to meet system needs.

3.4 SYSTEM SOFTWARE

3.4.1 Objectives and Approaches

In this section the design objectives of the SUMC multiprocessor operating system (MPOS) are presented, and the design approach selected to meet each of these objectives is described.

These design objectives are not mutually exclusive, and the basic features of the MPOS design were required to cover most, if not all of them. The following summary of MPOS characteristics represents a unified, coherent design approach that can be implemented for the intended environment.

3.4.1.1 Multiprocessing

- Objective

  MPOS must be inherently oriented to a multiprocessor configuration without requiring any effort on the part of its users to take full advantage of the potential for parallelism.

- Approach

All system components are treated as allocatable resources, especially CPUs and IOUs. Hardware interrupts, built-in resource interlocks (Lock-and-Go Instructions) plus appropriate operating system data structures support asynchronous operation of CPUs and IOUs. Moreover these mechanisms operate at either the firmware or operating system level, hence the user application programmer makes no special provisions in either his program design or its implementation for the multiprocessing capability.

3.4.1.2 Functional Modularity

- Objective

A high degree of functional decomposition must be achieved to obtain effective work load distribution, to minimize the effect of component failure (hardware and software), and to minimize the processing disruption caused when recovery procedures are instituted.

- Approach

The basic structural subdivision of MPOS into program management (MPOS/CPU) and data management (MPOS/IOU) mirrors the subdivision of the hardware into CPUs and IOUs. In addition, each of these MPOS subsystems is itself further decomposed functionally (see Fig. 3-21).

The usual penalty for strict functional modularity is the higher overhead for added execution time for intermodule communication and for added memory requirements of the modules themselves. In the SUMC design, hardware support for intermodule communication (e.g., multiple interrupt levels) substantially reduces this overhead. The penalty is further reduced by the unusually small number of levels in the hierarchy of program structures within MPOS. Reduction in levels is achieved by designing routines which are self-contained except for controlled sharing of data structures (tables and files).

- Interrupt and Timer Services
- Task Control for OS and User Tasks: task initiation, switching and termination
- Job Control: entry, selection, initiation and termination of user jobs
- Program Management: program load requests, program deletion, debugging and monitoring
- Data Handling: data read and write requests
- Virtual Memory Management: allocate and deallocate
- Diagnostics, fault isolation, recovery and reconfiguration
- Emulate S/360 CPU at OS/360 Control System Interface

CPU Functions

- Interrupt and Timer Services
- Task Control for IOU Tasks: task initiation, switching and termination
- I/O Dispatching and error recovery for data files, communications and displays
- Message Control, distribution and recovery, including polling and reconfiguration
- File Control, and file and record lockout, and file recovery
- Program and program overlay loading
- Spooling (buffering) for low speed peripherals
- Simulate S/360 selector and MUX channels

IOU Functions

Fig. 3-21. SUMC operating system functions.

3.4.1.3 Simplicity

- Objective

Reliability considerations dictate that MPOS be considerably less complex than its peer operating systems without sacrificing needed functions or capabilities. Complexity in this context is measured by:

1) Number of parameters at user interface
2) Number of interfaces internal to MPOS
3) Number of options open after system generation
4) Number of data structures
5) Simplicity of algorithms selected.

- Approach

Simplicity is achieved, in part, by focusing the design on the SUMC mission. MPOS does not have to be universal in the sense that OS/360 had to be equal to sustaining the sales and revenue of tens of thousands of computing systems in a variety of environments. While the SUMC mission is not as specialized as some fire-control and avionics missions, it is, nevertheless, a far cry from the extreme generality sought by manufacturers of general purpose computer systems.

Further simplicity is achieved through functional modularity and the selectivity built into the system generation process. When a specific version of MPOS is created from a library of all possible MPOS components, only those components necessary to support the mission need be included. It will often be advantageous to have different versions of the same component (e.g., Task Scheduler) to customize MPOS for particular operating environments.

3.4.1.4 Self Regulating

- Objective

MPOS must be unobtrusive and predictable in its behavior. MPOS must do its job with a minimum of required operator intervention and a minimum of surprises for its users. It must be human-engineered for its ultimate operating environment.

- Approach

To minimize operator intervention, MPOS will employ an internally stored contingency response file. Thus, for each error, conflict or information requirement which can arise in MPOS operation, there will be a default response or action. These defaults can be specified at the time MPOS is configured (via system generation programs) for a specific mission. This specification will be done parametrically, with no coding changes being needed. As a result, MPOS will be able to continue to operate even if the crew refuses to or is unable to respond to internal operating system problems. Thus a persistent error in a data transfer operation (main memory or peripheral device) will be corrected via error correcting code without requiring operator attention, although the error condition will be logged and, if traffic permits, displayed on an operator's console.

Human intervention will normally be required only to override mission defaults. When the crew requests this override mode, MPOS will include an operator's terminal in its decision making loop. When the system is not in override mode, the operator's terminal serves as a passive monitor.

3.4.1.5 Evolutionary

- Objective

MPOS must not be such a radical departure from current operating system technology as to risk failure in the initial development stage or to risk instability or poor performance in the operational stage.

- Approach

While the MPOS design includes several novel features, it represents a systematic attempt to improve on third generation operating systems by correcting for their faults rather than by introducing a new approach. The principal innovations are really matters of degree rather than kind.

Functional decomposition of operating systems using CPUs and IOUs has been successfully employed, e.g., UNIVAC LARC, CDC 6600. The novelty in the decomposition of MPOS into MPOS/CPU and MPOS/IOU is in the mechanization of the interface between the two, not the basic concept.

3.4.1.6 IBM S/360 Compatibility

- Objective

MPOS must support a high degree of compatibility with S/360 type, third-generation, computer hardware and software, in order to minimize the cost of transferring applications programs and files from S/360 to the SUMC environment. Pointless differences which artificially increase personnel training costs and errors must also be avoided.

- Approach

The ability to minimize the cost of conversion of S/360 programs for execution under MPOS on SUMC hardware will be accomplished by:

1) Stand-alone emulation of S/360 hardware
2) Source program conversion routines which rely on SUMC support of S/360 architecture and the high degree of functional similarity between OS/360 and MPOS.

3.4.1.7 Reliability

- Objective

It is essential that MPOS achieve as high a level of reliability as software technology will permit. Therefore a high level of confidence in the design integrity and operational stability must be assured in advance of each mission involving SUMC hardware and MPOS software.

- Approach

Reliability will be achieved by introducing a two-level operating system verification procedure. The first level is a design verification process which will systematically test each module in MPOS in its operating environment and, by suitable benchmark tests, will systematically overload typical hardware/software configurations to determine the operational limits of the MPOS/SUMC system.

The second level is highly mission oriented. Given the current state of the art in aerospace software, the most effective way to achieve confidence in the reliability of general purpose software is to run extensive, live operating tests, using actual mission hardware/software configurations and application programs.

To implement this two-level verification procedure, MPOS will include the following features:

a) Built-in monitoring capability with data logging

b) Data reduction and analysis routines

c) Ability to simulate interrupts and externally introduced data so as to permit "true to life" testing before live mission data and interrupts are available.

3.4.2 MPOS Data Structures

The data structures of the MPOS Executive determine legitimate activity within the system. The routines in the Executive which manipulate these data structures may come and go, but the data structures are permanent. They form the interface between components of the Executive in the same way that cables and pin-connectors form the interfaces between modules of complex electronic systems.

In practice, these data structures provide for much parameterization of Executive functions, e.g., type of support for prepaging, selection of tasks to be activated. Parameterizing reduces the number of steps and therefore instructions in the Executive routines which implement these functions and increases the ease with which a needed change in the Executive can be effected by changing parameter values instead of rewriting instructions.

The principal data structures (tables, lists and queues) of MPOS and their principal interrelationships are shown in Figs. 3-22 and 3-23. The fundamental control mechanism for both MPOS/CPU and MPOS/IOU is the Task Control Routine. A program is executed under control of MPOS by describing a basic unit of work called a task and then associating that program with the task. A task is described by means of a Task Control Block (TCB), shown in Fig. 3-24. A list of the TCBs of the tasks currently using CPUs and another list of the tasks currently using IOUs are maintained in main memory.

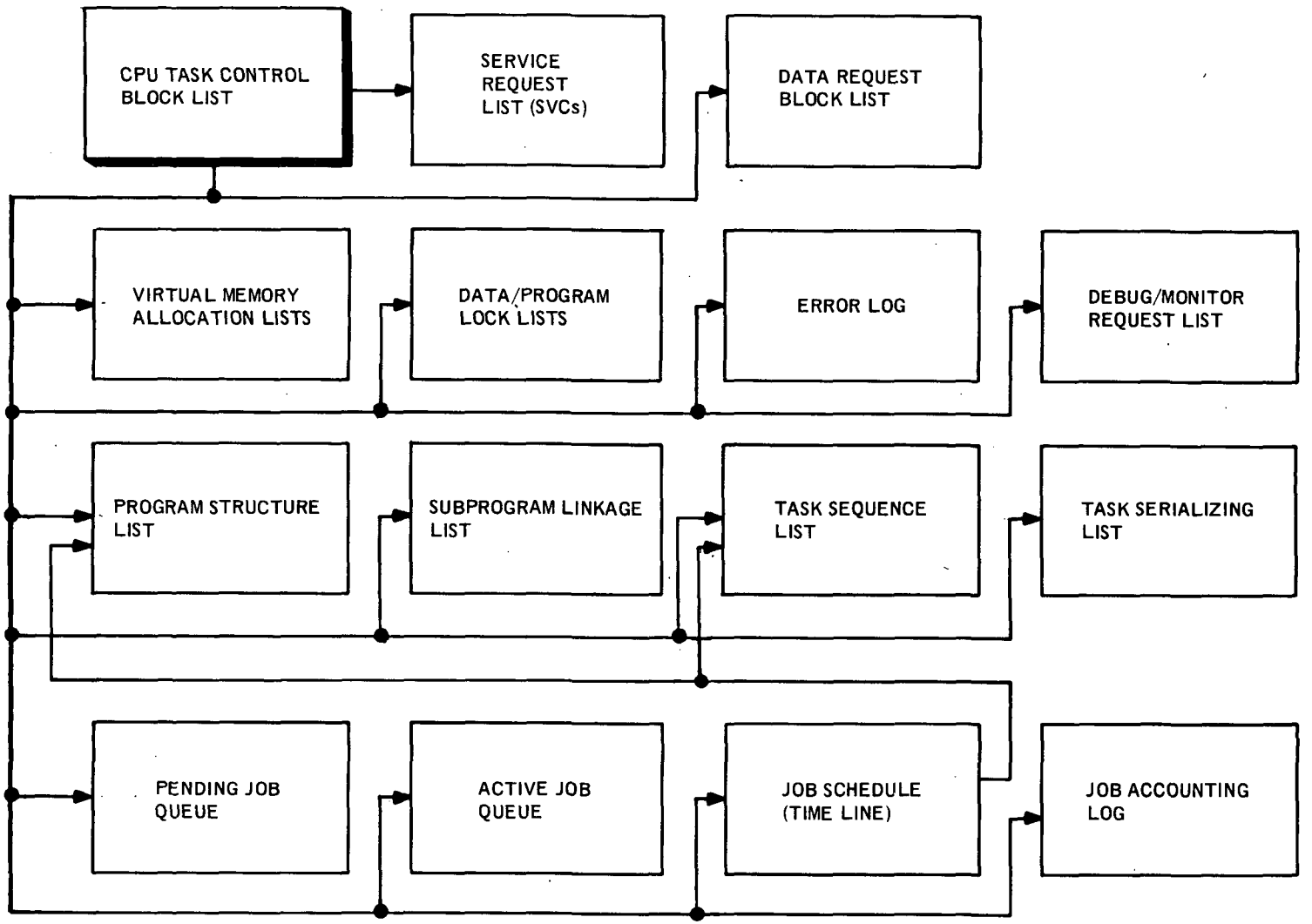


Fig. 3-22. MPOS/CPU data structures.

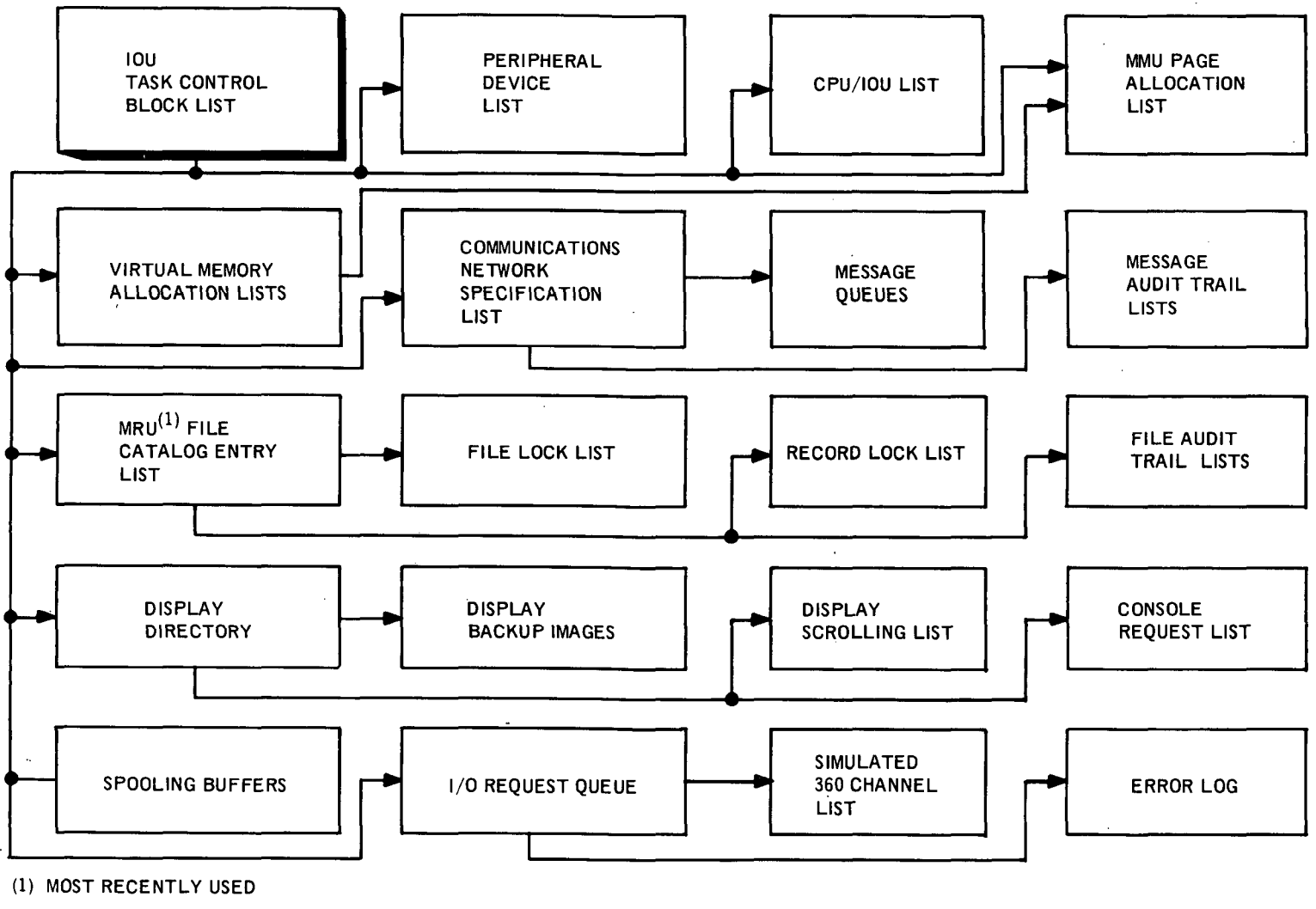


Fig. 3-23. MPOS/IOU data structures.

TASK CONTROL BLOCK
TCB LENGTH
POINTER TO NEXT TCB
TASK ID
CPU/IOU ID
TASK STATUS INDICATORS
LOCK STATUS (LOCK-AND-GO LINK OR NULL)
SWITCHING PRIORITY
ENTRY POINT ADDRESS
ENTRY POINT NAME
PROGRAM STRUCTURE LIST POINTER
JOB ID
REGION ADDRESS
REGION LENGTH
TASK STATE REGISTERS (PSW AND GENERAL REGISTERS)

Fig. 3-24. Task control block.

3.4.3 MPOS/CPU

The principal MPOS/CPU features are discussed in this section. The intent is to discuss the key components of the MPOS/CPU executive and their specific duties and responsibilities.

3.4.3.1 Interrupt Servicing and Processor Control

The multiprocessor organization will allow all processors (CPUs/IOUs) to be jointly and continuously sensitive to interrupts. Response to interrupts in the multiprocessor is controlled by the contents of the interrupt mask registers in each processor and by requiring that each processor, upon sensing the interrupt, reset its own interrupt flag and service that interrupt. Consequently, all SUMC processors (CPUs/IOUs) must contain interrupt analysis and control tasks.

Since the initial response to interrupts is handled in hardware/firmware, the processor servicing an interrupt can begin without making a memory (MMU) access.

When a memory access is eventually made, the coding and data accessed will be in nonpageable pages in memory (MMU). Therefore there is almost no overhead involved in allowing the several processors to share the burden of servicing global interrupts. However, some small amount of time may be lost by one processor if it responds to an interrupt just after another processor has. The second processor will immediately discover that the interrupt has been intercepted by another processor through the Lock-and-Go instruction mechanism and will return to its original task.

The multiprocessor organization requires that any processor unit be able to execute the reentrant interrupt servicing program for the interrupts that can be handled by its unit type. Thus, each processor has hardware/firmware interrupt handling logic and a ready interrupt servicing task associated with each interrupt level. The duplication adds little to the cost of the processor unit; and, since it must be interchangeable for system reconfiguration, the duplication improves reconfiguration support.

Thus, each processor unit contains a small, "bare bones" control program capable of sensing and responding to local interrupts. On the other hand, all other functions performed by the processors are scheduled by the MPOS executive which runs as a single task. Thus while global interrupt analysis is performed by whichever processor responds first, the initiation of tasks, scheduling of jobs and all other system functions are controlled by a unique executive task which can be performed by any processor but never by more than one at a time. Thus, there is always one job scheduler and one task scheduler in even the most elaborate SUMC configuration, and the CPUs and IOUs are treated as resources which are allocated by a single operating sytem.

To avoid excessive interruption of user tasks, the IOUs will normally service all external interrupts. Thus the multiprocessor can achieve the effect of insulating user tasks from interrupt interference that was sought after by CSC but without limiting interrupt servicing to one processor.. The IOUs can be designated as eligible to

respond to external interrupts and data bus servicing, while the CPUs are free to concentrate on execution of user tasks. In this way the IOUs can buffer the CPUs from external world events, thus allowing the tasks in the CPU to deal with the external world events at whatever response rate is appropriate.

3.4.3.2 Task Control

There are two types of tasks in MPOS: user tasks and system tasks. Both types are managed by Task Control and both are defined for MPOS by means of parameters assembled in a Task Control Block (TCB). A task is the smallest identifiable unit of work that can be processed by the MPOS Executive.

When a task is being processed, it is in one of four states as shown in Fig. 3-25. The task enters the system as Terminated and the transitions from Initiation to final Termination or Cancellation are determined by Task Control. Normally, i.e., without interruption or emergency, a task will proceed through the states as shown in Fig. 3-26. All transitions are summarized in Fig. 3-27 and are discussed in detail below.

To be made Ready, a task must have been allocated all of the resources it needs except a CPU and, if Pended to await an event, that event must have occurred. Being Ready means that the task has been assigned to a specific CPU, which assignment holds until Termination.

To be Activated a task must first be in the Ready state and then be selected for activation by the task scheduler.

The task scheduler is parameter driven, and it does not employ a fixed algorithm for selecting tasks to be activated. While no one task scheduling procedure is likely to be satisfactory for all SUMC missions, the following discussion shows the sensitivities of a baseline task scheduler.

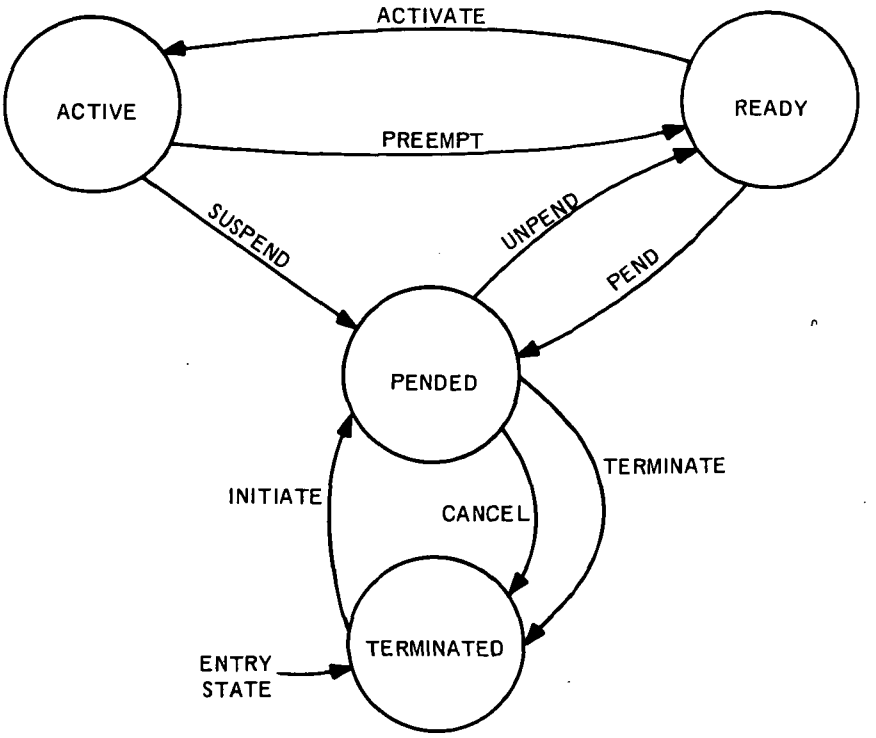


Fig. 3-25. State diagram for MPOS tasks.

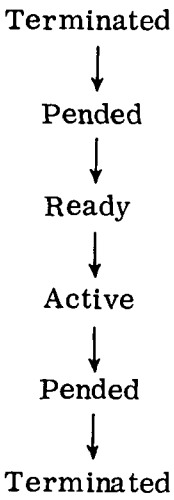


Fig. 3-26. Normal processing transitions.

Old Status →	New Status	Function	Description
Terminated →	Pended	Initiate	Place a nonactivated task in the queue of tasks to be made ready for Activation.
Ready →	Active	Activate	Give a waiting task control of a specific CPU (or IOU) and other resources allocated to it.
Active →	Pended	Suspend	Temporarily place an active task into a waiting state, at the request of the task, until a specified event occurs.
Active →	Ready	Preempt	Temporarily place an active task into a waiting state, but without consent of the task. The task can immediately compete for Activation.
Pended →	Terminated	Cancel	Forcibly remove a task from a Pended state.
Pended →	Terminated	Terminate	Remove a task from Pended state at its request, or the request of a controlling task.
Pended →	Ready	Unpend	Place a Pended task into contention for a CPU (or IOU), the reason for its being Pended having been removed.
Ready →	Pended	Pend	Temporarily place a ready, but waiting, task into a Pended state, involuntarily. Task can immediately compete for the Ready state.

Fig. 3-27. Task control transitional functions.

In general, task scheduling is deadline oriented (see Sec. 2.3.4). The guiding principle is getting jobs completed on time. The task scheduling strategy stresses throughput after deadlines. In fact it stresses throughput before equal sharing of system resources by competing tasks. As a consequence, when the task scheduler gains control after an interrupt for CPU_i has been processed (it always gets control at this point), will normally Activate (return control to) the same task that was active in CPU_i when the interrupt occurred. Thus MPOS tends to force an Active task to Terminate or Pend itself once it has been activated. As a consequence, high priority tasks are completed with minimum delay.

Activating a new task on a given CPU occurs only when a preemptive task becomes Ready or when the current task in that CPU Suspends itself -- usually either by issuing an I/O request or as a result of a page fault. The exception to this rule is that a task may be preempted if its estimated CPU time-to-complete is exceeded by X percent and then only when X is given as a task switching parameter.

A task can have two scheduling priorities associated with it:

- A scheduling priority, which determines the probability that it will be Initiated
- A switching priority, which, for a Ready task, determines the probability that it will be Activated.

These two priorities are, by definition, independent of each other, although a task with high scheduling priority may also, coincidently, have high switching priority.

When emergency conditions arise, when a deadline is in jeopardy or when the tasks priority dictates, a task will be given exclusive use of the CPU, i.e., no other user tasks will be given control of that CPU until the task in exclusive use Terminates or Pends itself.

Time-slicing will not be employed as a basic task scheduling strategy but will be used only as a last resort when human-engineering principles dictate smoothing response time fluctuations for tasks which interact directly with a manned terminal.

A spare task slot (a position in the list of Ready tasks) is reserved for emergency use, i.e., for scheduling tasks which have preemptive priority. To avoid losing this task slot when emergencies cause cascaded scheduling of such tasks, a lower-priority, Ready task will be Pended, if necessary, in order to reduce the number of Ready tasks (i.e., tasks contending for use of a CPU) to an acceptable level consistent with the nature of the emergency.

Actual switching of tasks by the task scheduler is very rapid in comparison to task switching times for typical S/360 style systems. The reasons are that, in SUMC, most of the housekeeping functions saving and restoring of a task's environment, are done in firmware and there is extra hardware for each "permanent" system task to have its own scratchpad area for general registers, PSW, etc.

3.4.3.3 Job Control

From the user's point of view, getting the MPOS/SUMC system to do something for him requires that he submit a job. A job typically is composed of explicitly defined user tasks, and each task may call for the execution of one or more user-developed program modules. When the job is run under control of MPOS, other MPOS tasks will be invoked to respond to requests for service embedded in the user defined tasks, e.g., the task invoked to load a user program module. Thus execution of a job under MPOS control involves the performance of many tasks, some explicitly defined by the user, others implicit in the service requests and resource demands made by the user tasks.

Resource allocation is a paramount consideration in job scheduling in a multiprogramming environment, and memory management strategies are a paramount consideration in resource allocation. Thus memory management strategies for MPOS are chosen to harmonize with and to enhance the job and task scheduling strategies.

Jobs to be run by MPOS are entered into job queues, which consist of lists of job files (see Sec. 3.4.3.9) describing the jobs to be run. The principal queue of jobs in the MPOS environment is the queue that is dynamically obtained by examining the time-line schedule. Other jobs are entered into job queues as a result of action taken by other jobs and explicit requests entered by crew members. In fact, one of the jobs that can be entered on a nonscheduled basis is a job whose function it is to revise the time-line schedule to meet new constraints based on inputs furnished by crew members.

The job scheduling strategy for MPOS will be based on satisfying the following constraints, given in order of decreasing priority:

a) Conformance to time-line schedules. This requires the scheduler to compare the estimated time-to-complete for a job with the due-out-time as specified by the time-line schedule.

1) For an unqueued (nonready) job this means providing for its resource needs and than placing it in a ready queue when current-time plus estimated-time-to-complete is within a specified tolerance of due-out-time.

2) For a ready job this means increasing its execution priority when there is a danger that due-out-time will not be met unless the job is activated within a specified time interval. In some cases this may result in low priority jobs being suspended or canceled in order to improve the nearly-late jobs chances of contending successfully for the resources it needs.

3) For an active job this may mean canceling or suspending some tasks belonging to low priority jobs, and/or it may also require increasing the switching priority of a task in order to improve its chances of competing successfully for the CPU.

b) Conformance to a job scheduling priority initially assigned independently of consideration for due-out-time requirements. This assignment allows for the case in which, for example, a non-time-line job of considerable urgency is submitted for execution.

c) Efficient use of system resources. When, by measurement of suitable parameters, MPOS determines that one or more of the systems resources (e.g., a CPU) is idle, it will then run the Job Scheduler in an effort to get a job into the ready queue that can make use of those resources. An effective way to use resources is to give MPOS a large and diverse list of very low priority jobs which can be used to supplement more important work whenever system resource utilization drops below an acceptable level.

3.4.3.4 Program Management

Programs are stored in the form of data files and are accessed by the MPOS Loader through the use of the file management facilities of MPOS/IOU. Program data files, as are other data files, are accessed by MPOS/IOU through the system file catalog.

The program module containing the current entry point for the task being initiated is automatically loaded, if necessary, during task initiation. A separate, distinct service request to the MPOS Loader is not required. If a copy of that program module is currently loaded and in a reusable state, no loading is done.

The shared program facility in MPOS allows two or more tasks running in any of the CPU/IOUs to share the same copy of a reentrant program module. A reentrant program is one which does not modify itself during execution and which has no modifiable data storage areas of its own: All input data, work areas and output data areas are provided by the calling program. When the MPOS Loader is given a request to load a reentrant program, it first checks to see whether a copy of that program has already been loaded. If it has, the physical memory locations occupied by that copy are entered into the appropriate entries in the requesting task's address translation table. Due to the use of virtual memory management, the same reference to a given program may appear in different sections of the virtual address space of each task sharing the common code, even though it occupies a unique section of the MMU.

3.4.3.5 Data Request Interface with IOU

In the CPU/IOU partitioning for SUMC, all of the file processing, display support and peripheral device control is done in the IOU. User and system tasks in the CPU issue requests for data transfers at the logical record level. Such requests are typically parameterized as follows:

- File ID (name of data file)
- Record ID (name or number)
- Passwords (security control codes)
- Function (read, write, replace, delete)
- Address of record area
- Length of record area.

This information is passed to an IOU by means of the following procedure:

- The parameters of the request are entered into a Data Request Block, which is then linked to a list of such blocks.
- The CPU-to-IOU mailbox entry for the task issuing the request is modified to show that an outstanding request is present.

When the request has been satisfied by the responding IOU, the IOU-to-CPU mailbox entry for that task will be modified to show that a data request, specifically the one which caused that task to be suspended, has been satisfied (see Sec. 4.4, Interprocessor Communication).

All other file management processing is done in the IOUs under control of MPOS/IOU.

3.4.3.6 Resource Allocation and Protection

In MPOS the following are the types of allocatable resources:

- Processors: CPUs, IOUs
- Task slots (interrupt levels) in a CPU or IOU (control points)
- Memory: page frames in MMU, page frames in Auxiliary Memory
- Programs and program files
- Data files and data records
- Input/output devices and their controllers
- Logical I/O channels (i.e., data bus priority groups).

These resources are managed by MPOS using a coherent resource allocation strategy that is applied to the management of each of these resources. Any specific one of these resource types can be dedicated (preallocated) to a job or a task in the interests of meeting deadlines or improving throughput. For example, a task slot in at least one IOU is permanently dedicated to the task which services MMU page faults. In addition, that task has permanently resolved for it the following: sufficient MMU page frames to guarantee that all of the programs needed in the execution of the task are resident. The reasoning behind this is fairly obvious: page faults will happen often enough that, in spite of the low access time expected for Auxiliary Memory, it should be made impossible for the servicing of a page fault to itself cause a page fault. Moreover, the dedicated task slot guarantees that the page fault servicing task will never have to wait for another task to terminate (give up a task slot) in order to allow it to contend for IOU time.

Avoiding resource contention is important to throughput and meeting deadlines. MPOS will avoid contention by:

a) Task Scheduling Strategies

Reserving sufficient resources for the sum of all critical tasks and/or reserving less than that and being prepared to preempt additional resources from less critical tasks as the reserves are used.

For noncritical tasks reserve the maximum, or near maximum, of the task's resource needs just prior to placing the task in the Ready queue. For MMU resources, this could be accomplished by partitioning main memory instead of employing variable regions).

Estimating the memory resources needed for a given job mix can be done using the following formula:

M = page size x average number of tasks x average number of pages/task

b) Reducing Paging Overhead

Various tactics for reserving or dedicating MMU page frames can be readily employed through varying the prepaging parameters (i.e., using predicted page usage to anticipate page needs). The tactical gain resulting from such a parameter change can be startling. The following elements form a prepaging vector, which could be used to control resource use and task completion time:

- Extent of prepaging

  No pages prepaged

  Only the P-counter page

  Only the running set (r, where $r < w$)

  Only the working set (w)

  The full set (f, where $f \geq w$)

- Number of pages preempted by prepaging
- Number of pages in full set.

Further use of prior information is attainable by assigning a "sticking power" to pages of high frequency tasks, which would alter the page replacement pattern by reducing the probability of replacement of pages with "high" sticking power.

c) Sharing of Reentrant Programs

In effect, an arbitrary number of tasks can share a single copy of a reentrant program module. This is mechanized through the program structure list for each task. The entry for a sharable program is marked so as to cause the loader to check for an existing, loaded copy before loading another copy.

Avoiding system deadlock is crucial to the success of MPOS/SUMC. The following strategies are applicable:

a) Limit cases in which a task is both queued for a resource and is retaining exclusive control of other resources. Such limitation can be done empirically by:

   1) Enforcing programming standards to prevent deadlock unless it is unavoidable.

   2) Detect this case during program tests and/or mission simulation and attempt program revision to correct for this behavior.

   3) As a last resort, add the names of such tasks to the task serializing list (a list of those tasks whose mutual contention for resources is known to allow deadlocks to occur).

b) Backing out of deadlocks involving "bulk" resources

   For main memory deadlocks the task scheduler can instruct memory management to page the least recently used pages of active tasks to auxiliary memory. If that is not sufficient, the task scheduler can then request that a preemptive replacement algorithm be executed which pages some or all of the pages of active tasks of lower priority than the two deadlocked tasks. As a last resort, the task scheduler can suspend or cancel one or more lower priority tasks. This same approach can be used for auxiliary memory, even though the penalty there is that the read-only pages of the lower priority task will have to be reloaded before that task can be reactivated.

3.4.3.7 Diagnostics, Fault Isolation, Recovery, and Reconfiguration

The functional testing of SUMC hardware components (such as input/output devices) and the partial testing of memory units and processors can be accomplished through the use of software diagnostic routines. More detailed self-checking of CPUs, IOUs and MMUs can best be done via microprogrammed diagnostic routines as is being done on IBM's System 370 computers. At the microprogram level, all pertinent flip-flops, registers and intermediate results are accessible.

MMU Error Recovery is handled in MPOS as follows:

1) Read-only pages

IOU is given a request to reread the program or data page into a different physical page frame in the MMU and, if successful, to modify the ATU entry accordingly.

2) Read-write pages

Critical program pages, data pages and executive data structures will be supported with backup copies in alternate MMU page frames. If an error occurs for a writable page that has been modified and does not have a backup copy, then the affected task is canceled and reinitiated.

On-line diagnostics and fault isolation tasks can be initiated manually from control terminals, or automatically by MPOS upon detection of faults. A faulty unit (MMU, CPU or IOU) can be "dropped out" of a SUMC configuration automatically, if it is not the last such unit. Multiple unit failures may require manual intervention for reconfiguration.

3.4.3.8 Emulation of IBM S/360 CPU

While most of the S/360 architecture is either directly supported in the SUMC architecture or (in Selector and MUX channels) simulated by MPOS/IOU, there may be additional S/360 features (e.g., interval timers, S/370 instructions, CPU serial numbers, S/370 virtual memory) which must also be supported in order to achieve a required level of S/360 emulation. In many cases these additional features will be implemented via SUMC microprograms, but there may be some residual features whose simulation by MPOS/CPU software represents the best tradeoff. An example would be the servicing analysis and classification of S/360 Program Controlled Interrupts.

3.4.3.9 Program and Job File Maintenance

Utility programs which operate under control of MPOS provide facilities for creating, updating and deleting program files and job files. All such files are

cataloged data files and are manipulated by using standard MPOS/IOU file control routines. Program files are also called program library files, emphasizing that such files typically contain from 10 to 100 related programs or program modules. Job files are also called job library files to emphasize that such files typically contain from 2 to 20 related job descriptions.

In addition to creating, updating and deleting entire program modules, these file maintenance routines provide for the addition, deletion and reordering of both individual lines of coding within a source program module and individual job control statements within a job description.

3.4.3.10 Programming Language Support

Of the programming languages in wide use today the following are considered to be most appropriate for program development in the MPOS/SUMC environment.

a) JOVIAL (J6)
b) PL/1
c) FORTRAN (ANSI approved)
d) COBOL (ANSI approved)
e) APL

Suitable compilers for these and other applications-oriented languages or their equals should be supplied as part of the MPOS program preparation facilities.

Compilers, in executable form, will be stored in system program library files and can be executed using standard job control procedures.

3.4.3.11 System Generation

System generation is the process by means of which a customized version of MPOS is created in conformance with a specified set of system functions and system

performance criteria. System generation procedures are carried out entirely under program control on a standard, system-generation-oriented SUMC configuration by using a standard version of MPOS. There are two modes of operation for this process:

a) Initial system generation where no previous customized version of MPOS exists.

b) Updating an existing version of MPOS by systematically replacing program modules and regenerating selected MPOS data structures.

The following specific functions are performed for an initial system generation process.

a) Creating the appropriate data structures to support the SUMC hardware and software configuration described by the input data presented to the system generation process, including:
   1) Creation of a physical device list
   2) Creation of a data file catalog
   3) Formation of an initial main memory allocation for all resident MPOS routines, tasks and lists.
   4) Specification of initial job queue entries.
   5) Creation of an initial Task Control Block List and related task management lists.
   6) Creation of initial MMU ATU translate tables.
   7) Creation of MMU Page Allocation List.

b) Creating the MPOS system program library files by selecting precompiled system program elements from a master program library.

c) Running system validation tests by using the newly generated or updated versions of MPOS.

3.4.4 MPOS/IOU

The salient features of MPOS/IOU are discussed in this section. The major routines of MPOS/IOU are listed here along with a discussion of their capabilities and responsibilities.

3.4.4.1 Interrupt and Timer Services

Servicing IOU interrupts is done under control of MPOS/IOU by using the same interrupt analysis and control programs employed by MPOS/CPU. Thus any IOU can, and in emergencies must, be able to service any global interrupt by using the interrupt servicing routines for those interrupts not masked or inhibited for IOUs. One or more IOUs in the multiprocessor configuration are usually connected to the main data bus with auxiliary memory interrupts permitted. This arrangement assures that pacing interrupts will be serviced by the first available IOU as long as there is at least one IOU in operation.

3.4.4.2 Physical Memory Management

Physical memory management (translation of virtual addresses to physical addresses in MMU and auxiliary memory) for CPUs and IOUs is by hardware and microcode. The only aspect of physical memory management visible to MPOS/CPU software is the ability to determine saturation levels in MMUs and auxiliary memory. Thus MPOS/IOU will have access to page frame usage counts for both the MMUs and auxiliary memory. MPOS/IOU will pass this information to MPOS/CPU resource allocation routines via MMU for use in measuring system resource utilization and preventing system overload.

The MPOS/IOU tasks which service MMU page faults are the only tasks which are directly involved in physical memory management. These tasks execute MPOS/SUMC page placement and page replacement algorithms. Consequently, these same

MPOS/IOU tasks also maintain the MMU Page Allocation List and the Auxiliary Memory Page Allocation List. (see Sec. 3.4.3.6).

3.4.4.3 Task Control

MPOS/IOU will use the same task control routines used in MPOS/CPU, the principal difference being that the available task slots in the IOUs are normally dedicated to the execution of specific IOU functions while CPU task control manages the multiplexing of user tasks together with MPOS/CPU tasks. Thus, for a given MPOS/SUMC configuration, the MPOS/IOU task structure is considerably more stable than the MPOS/CPU task structure.

3.4.4.4 I/O Device Control

MPOS device control routines initiate data transfer and control command transfers for all devices attached to the main data bus. I/O interrupt control routines are assigned to servicing all external interrupts received from device controllers attached to the bus. MPOS device control performs the following functions:

- Accepts I/O requests from MPOS user and system tasks.
- Queues I/O requests for dispatching. Queuing is by device or controller, whichever is appropriate.
- Dispatches I/O requests on a FIFO basis within task priority, by sending appropriate command sequences to the device controller.
- Transfers data between the main data bus and preallocated I/O buffer areas in an MMU.
- Performs I/O error recovery procedures as required.
- Requests initiation of device diagnostic and fault isolation tasks as required.
- Enters device error conditions into the system log.
- Notifies MPOS file control when an unrecoverable error from device, controller or bus requires a file-oriented, rather than device-oriented, recovery procedure.

3.4.4.5 Program Loading

Object programs produced by compilers from source programs are stored in Object Program Library Files, which are implemented by means of standard, cataloged, data files residing in the auxiliary memory. The MPOS Loader needs three items of information to satisfy a program load request:

- Name of program library file containing the program.
- Name of program (or program module) to be loaded from the specified program library file.
- Name (address) of the Task Control Block (TCB) of the task for which the loader request is being executed. The TCB contains pointers to other MPOS data structures which are in turn manipulated by the loader. These structures are:

  1) Virtual memory allocation lists
  2) Program structure list
  3) Subprogram linkage list.

The loader uses the normal MPOS/IOU file control facilities to access the data files containing program libraries and uses the normal MPOS/CPU Virtual Memory Allocation routines to acquire the virtual memory address space for the program being loaded. Indirectly, the loader uses these routines to acquire the MMU allocation routines for the necessary physical memory space in MMUs and auxiliary memory.

Symbolic name references and definitions generally appear in the object program modules produced by compilers. These names describe intermodule referencing in object modules. The loader uses these names to link references during the loading of modules so that the loaded program has name references replaced with:

- The virtual addresses
- A pointer to the entry for that name in the Program Structure List (PSL). A reference to such a pointer appears in the form of an MPOS loader service request plus the pointer to the PSL.

The loader will resolve references by name in either of the two modes described, according to the information supplied in the Program Structure Table.

3.4.4.6 Message Control

MPOS Message Control Routines initiate message transmission, routing and logging for all message traffic in the multiprocessor. Message Control Routines call on MPOS/IOU Device Control to handle the actual device control functions. MPOS Message Control performs:

- Message header analysis
- Date/time stamping
- Assignment of routing codes and queues for output messages
- Management of routing queues and alternate terminal assignments
- Management of message journals and message retrieval, including "lost message" service
- Initiation of terminal testing and diagnostic tasks
- Control of bulk storage for direct buffering of messages and routing queues.

An extremely important feature is input validation for each input message. Validation consists of the following basic steps, which are executed before the message is available for processing:

- Header validation and analysis
- Message format validation
- Validation of fields in the message body for correctness of content (e.g., range check on numeric fields).

3.4.4.7 File Control

One of the principal objectives of MPOS File Control is to provide a comprehensive file management facility that offers the user both a device independent file

processing interface and access techniques which are independent of the physical arrangements of data on the storage device, i.e., independent of the storage structure employed. In this context, device independence means that the system is able to make device substitutions (e.g., disk for tape) without requiring alteration in the user program or its interface with MPOS File Control.

Storage structure independence guarantees that user programs and their File Control interfaces can remain fixed even when the track capacity of a disk or the block capacity of a BORAM device is changed. Design for independence allows introduction of new storage technology into the MPOS environment with software changes limited to the MPOS File Control routine.

A secondary objective is to manage the automatic allocation and deallocation of peripheral devices (displays, backing store devices, terminals, etc.) and the automatic mounting and dismounting of storage media for removable media devices.

In MPOS the user accesses data one logical record at a time. The interface with MPOS File Control is at the logical record level with the user specifying a retrieval key value or record number whenever he wishes to read or write a logical record.

The following pages describe the MPOS File Control facilities implementing these concepts.

MPOS maintains an on-line directory of all files in the system, including user data files, program library files, system scratch files and job files. This directory is a semipermanent repository of information about files and, as such, it relieves the user of the need to supply file description data other than the catalog

name each time the file is accessed. File description parameters maintained in a typical file catalog are:

- File identification (file-name)
- File access passwords (read, write, delete)
- File sharing criteria
- File label creation and checking information
- Identification of backup copy or alternate file
- Physical identification of file storage
- History of file usage and current disposition
- Physical device requirements and mounting instructions
- File structure and names of the file control program needed for processing the file
- Date of creation of catalog entry
- Type of file backup and recovery procedures to be invoked for this file in the event of an unrecoverable error.

Using the catalog, information pertaining to a file can be entered, deleted or retrieved with only the name of the file as a symbolic pointer to the corresponding file catalog record.

Unlike some current file systems in which data sharing and data interlocking are controlled by cooperation between the users of the data, the MPOS system builds these functions into the file management subsystem itself. Thus, data sharing and interlocking do not depend on cooperative efforts of all the tasks which choose to access a specific item of data, but rather it is centrally controlled by the user who cataloged the file. Consequently, a failure in data sharing or interlocking can occur only through hardware or logic error in file management software/firmware. This arrangement is superior to depending on the many tasks using a given data file

to control sharing and interlocking cooperatively since a single program error in one of these tasks jeopardizes the intention of the sharing and interlocking. Moreover, since sharing and interlocking is under control of a single program, changes in the status of data for sharing can be effected through changes in the file control parameters which are centrally managed by the file management subsystem. Improved reliability and control may be obtained through functional decomposition; i.e., file protection and control responsibility are not shared between applications, programs and MPOS file control.

Data file protection and protection against device misuse are further assured in the multiprocessor by the division of labor between CPUs and IOUs. User tasks in a CPU can gain access to data only by data requests which are serviced by an IOU. Also, no user task can be performed in an IOU unless it has been converted into a system task and is scheduled as a system task by MPOS.

Unlike the decentralized approach to I/O control, MPOS will, in effect, stand between the user and system resources including files and peripheral devices. Special routines developed to handle special purpose I/O devices must be designed and built to MPOS system routine specifications and standards and then incorporated into the file management subsystem of MPOS.

MPOS File Control ensures basic file organizations or data structures. Users are expected to map each of their data structures (data files) onto the most appropriate MPOS file organization. The organizations are:

- Sequential -- a named linear array of records
- Partitioned (modular) -- an ordered set of named linear arrays or records
- Indexed (named) -- an ordered set of named records.

Within each organization, logical records can be stored and retrieved:

- Sequentially, e.g., read next record, write next record
- By record number, e.g., read ith record, write jth record
- By record name, e.g., read record A, write record B.

Table 3-8 shows permissible combinations.

TABLE 3-8. COMBINATIONS OF ACCESS MODES

Organization	Access Modes for Logical Records
Sequential	1) Sequential (next) 2) By record number
Partitioned	1) Sequential within partitions 2) By record number within partitions (selection of partition is by partition name)
Indexed	1) Sequential 2) By record number 3) By record name

Partitioned organization is used primarily for program and job library files. Sequential and indexed organizations are classical file structures.

3.4.4.8 Spooling

Spooling denotes the buffering of data going to and from low-speed input output devices. Spooling obviates system-wide slow-downs by creating intermediate files, usually in bulk storage, which allow processing programs to run at full speed. Then, through the use of special system routines, these intermediate files are written to lower speed output devices. In MPOS, spooling for input and output files is controlled by MPOS/IOU.

In MPOS device independence is obtained as an additional benefit of spooling. A file spooled to disk could be rerouted for display or storage from one terminal device to another without change in the program that originally created the file.

Files spooled to disk from low-speed input devices are cataloged and held until they can be accessed by the intended user or system job. Files spooled to disk by a user or system job and destined to be written to output devices are automatically scheduled for transcription by MPOS without explicit operator action.

3.4.4.9 Simulation of IBM S/360 Selector and MUX Channels

To support emulation of S/360 hardware, certain MPOS/IOU routines will be written to interpret IBM S/360 physical I/O functions. The principal features are:

1) Trapping of S/360 privileged I/O instructions (e.g., start I/O)
2) Interpretation of S/360 channel programs (CCW chains) and translation of these channel programs into equivalent SUMC-IOU/data-bus operations.
3) Fabrication of S/360 sense-byte indicators and error indicators.
4) Simulation of S/360 channel-to-CPU interrupts.

The routines performing these functions will run under control of MPOS/IOU. Emulation of S/360 hardware and execution of IBM software require that IBM peripheral devices, and in some cases their controllers, be attached to the data bus of the multiprocessor configuration on which emulation is to take place

3.4.5 S/360 Interface Levels

3.4.5.1 Source Language Conversion

Programs developed for compilation and execution on IBM S/360s can be converted to execute under control of MPOS in the SUMC environment. It is

estimated that conversion costs associated with transferring S/360 programs to SUMC environments will be held to less than 10 percent of the original development costs of those programs. SUMC and MPOS are convertible because:

- SUMC includes the S/360 universal instruction set as a subset of its own instruction set.
- MPOS will support a user program interface that is functionally similar to OS/360 user program interface. The execution environments for MPOS object programs will, by design, accommodate the program management and data management capabilities of OS/360.
- MPOS compilers will be able to recompile S/360 programs, thereby producing object coding that interfaces with MPOS to achieve the operating system support originally provided by OS/360.

The conversion envisioned will apply more or less effectively to S/360, programs according to the programming languages used in their original development. Assuming that COBOL, FORTRAN and assembly languages are supported with appropriate compilers, library routines and linkage editors, the figures in Table 3-9. show the relative cost of conversion. PL/1 and JOVIAL have been included to illustrate the range of conversion costs associated with higher level languages.

Programs developed to run on IBM S/370 computers under AOS (a new IBM operating system which is likely to be announced in 1972) should be convertible to MPOS at even lower costs if the expected implementation independency of AOS is achieved, i.e., if AOS provides a cleaner interface for user programs than OS/360.

Depending on the number of S/360 programs likely to be converted to run under MPOS, the following conversion aids would be cost-effective. They are listed in order of decreasing usefulness:

1) Job Control Language translation aid (OS/360 to MPOS job control language)
2) Program library translation aid
3) Data File conversion routines
4) Linkage Editor Control statement translator (OS/360 to MPOS program structure description language).

TABLE 3-9. S/360 TO SUMC PROGRAM CONVERSION COSTS BY PROGRAMMING LANGUAGE

Programming Languages Used	Relative Conversion Cost*
OS/360 FORTRAN G-H	1-4%
DOS/360 FORTRAN F-G	2-6%
OS/360 COBOL ANS	1-2%
DOS/360 COBOL ANS	1-2%
OS/360 COBOL E and F	1-5%
DOS/360 COBOL D with no DAM files	1-3%
DOS/360 COBOL D with DAM files	2-10%
OS/360 PL/1s	10-20%
DOS/360 PL/1s	15-25%
JOVIAL (J6)	25-40%
S/360 Assembler (with no PIOCS, no BTAM and no QTAM)	35-50%
S/360 Assembler (with PIOCS or BTAM or QTAM	60-90%

* As a percentage of original development cost.

3.4.5.2 Direct Execution of S/360 Programs

On isolated stand-alone SUMC configurations (either single CPU or multiple CPU) it will be possible to emulate IBM S/360 or S/370 hardware with the accuracy needed to run S/360 applications programs under control of OS/360, OS/370 or their successors. Such emulation is based on the inclusion of the S/360 and S/370 universal instruction sets with emulation of privileged instructions as subsets of the SUMC CPU instruction set.

Considerations of system reliability and performance preclude both concurrent simulation of OS/360 with MPOS and hierarchical simulation of OS/360 by MPOS; the possible risk to space station life support systems is too great.

The stand-alone emulation of S/360s on dedicated SUMC configurations is feasible provided that S/360 peripherals or their replacements can be operated satisfactorily by S/360 emulation programs running in IOUs. These programs would intercept S/360 I/O instructions, interpret S/360 channel programs and simulate S/360 selector and MUX channel interfaces.

Section 4.0

MULTIPROCESSOR OPERATION

Rather than attempt a complete description of how the proposed multiprocessor is to operate, certain aspects of the operation have been selected for description:

- Dynamic task management
- I/O subchannel servicing
- Paging operations
- Interprocessor communication
- System state transition control
- Failure response.

Treatment of each aspect is broad enough to engage with the other aspects and to reference previous structural descriptions so that a unified (if not uniform) picture of entire system operation emerges. Areas less well-defined are among those recommended in Sec. 5 for further study.

4.1 DYNAMIC TASK MANAGEMENT

A task in the multiprocessing system must be in one of four processing states -- Terminated, Pended, Ready, Active -- as described in Sec. 3.4.3.2. Proper transition from one state to the next constitutes dynamic task management and is described in detail below.

4.1.1 Initiation

Task initiation is one of the functions of the Task Control Routine in the MPOS Executive. Tasks are initiated in both CPUs and IOUs. The fundamental process is

that of changing a task's status from Terminated to Ready. A new task that has never before been made Ready must start as a Terminated Task; i.e., it holds no system resources in its name.

The change from Terminated to Ready includes changing task status to the intermediate state of Pended where it waits for a resource to be allocated, or other event(s) that must occur before the task can be given control of a CPU or IOU. When all such requirements are met, including the acquisition of a Task Slot, the task's status is changed from Pended to Ready. More specifically, the operations involved in this transition are shown in Fig. 4-1.

4.1.2 Activation

Task activation is one of the functions of the Task Control Routine in the MPOS Executive. Tasks are activated in both the CPUs and IOUs. The fundamental process is that of changing a task's status from Ready to Active. An Active task is one which has control of a CPU or an IOU. To be eligible for activation a task must have been assigned a CPU/IOU Task Slot and all other resources (see Fig. 4-2) needed to run it except use of the CPU/IOU. The transition can be made (Exit 1) only if system resource checks are satisfied; otherwise, the task remains Ready (Exits 2-5) from which it may become Pended and Cancelled. When Ready, a task awaits only the assignment of a CPU/IOU.

4.1.3 Termination

Task Termination is a function of the Task Control Routine in the MPOS Executive. A task can be Terminated either by inherent request (Termination) or by imposed request (Cancellation). Figure 4-3 shows the successive procedures and checks made for proper Termination (Exit 1). It is improper to Cancel a Terminated task (Exit 2). To be Terminated, a Ready or Active task must first be Pended. A Pended task is neither in control of, nor contending for control of, a CPU/IOU. Task Termination is essentially a process of changing a task's status from Pended to Terminated. Upon

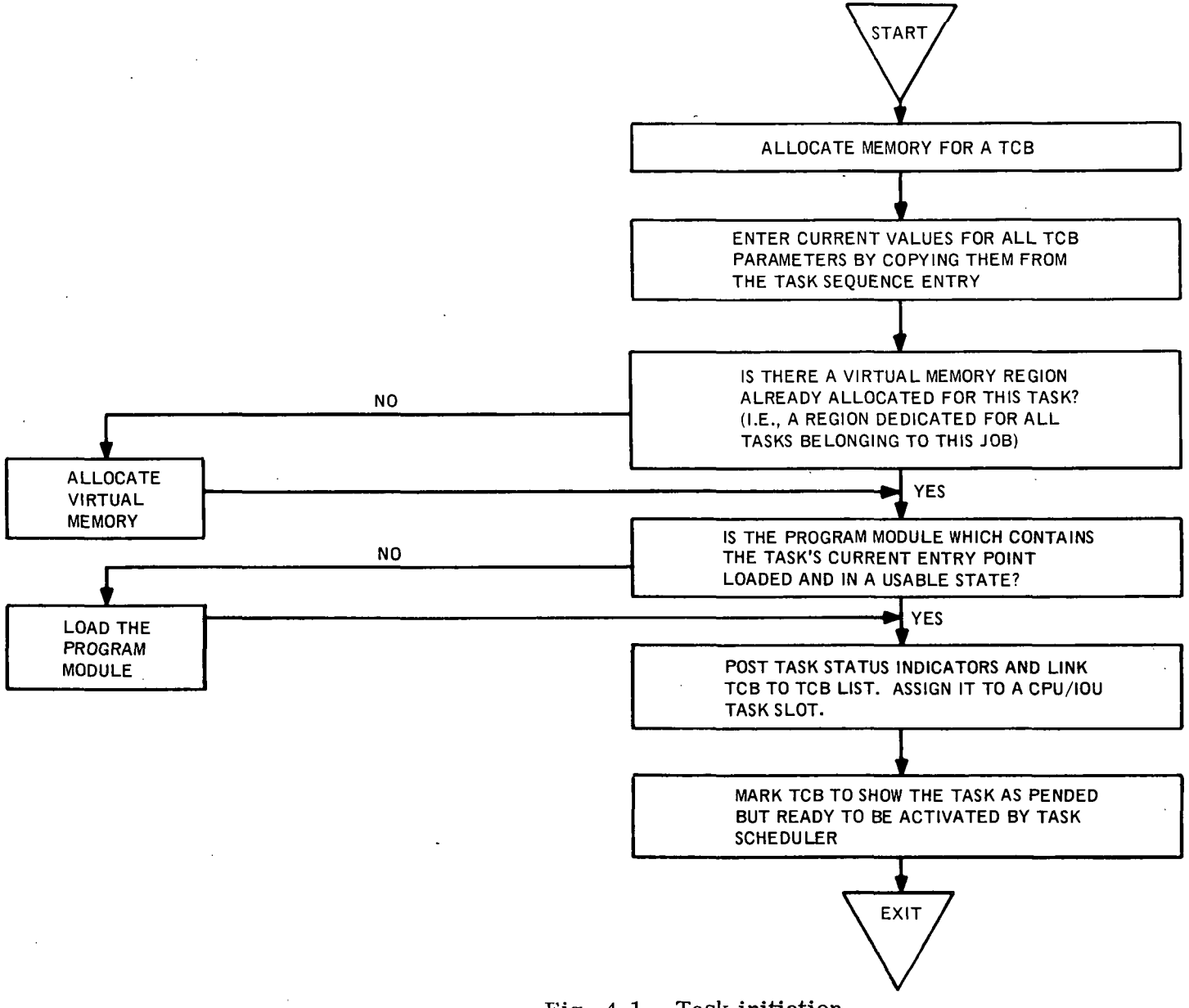


Fig. 4-1. Task initiation.

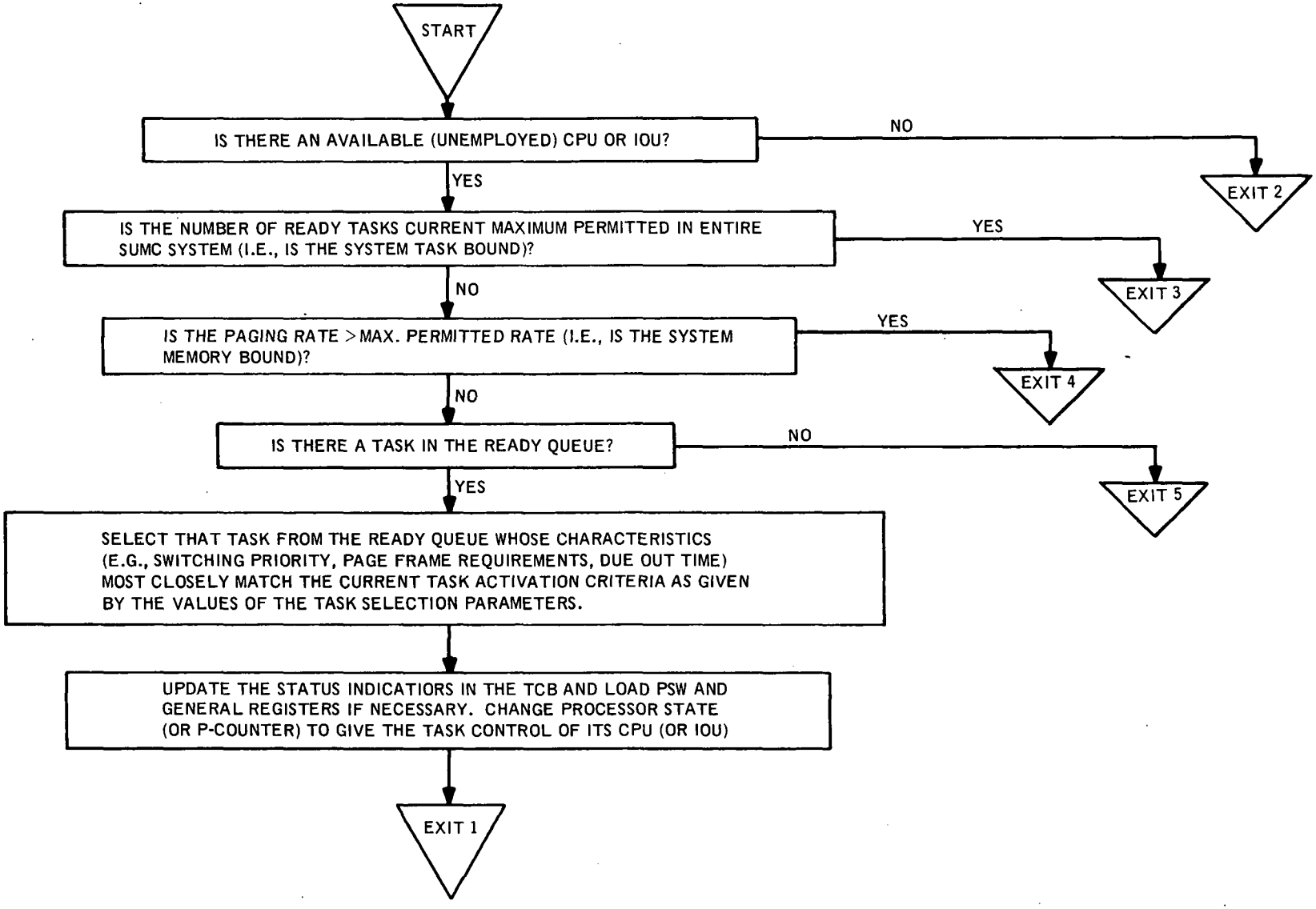


Fig. 4-2. Task activation for CPU or IOU

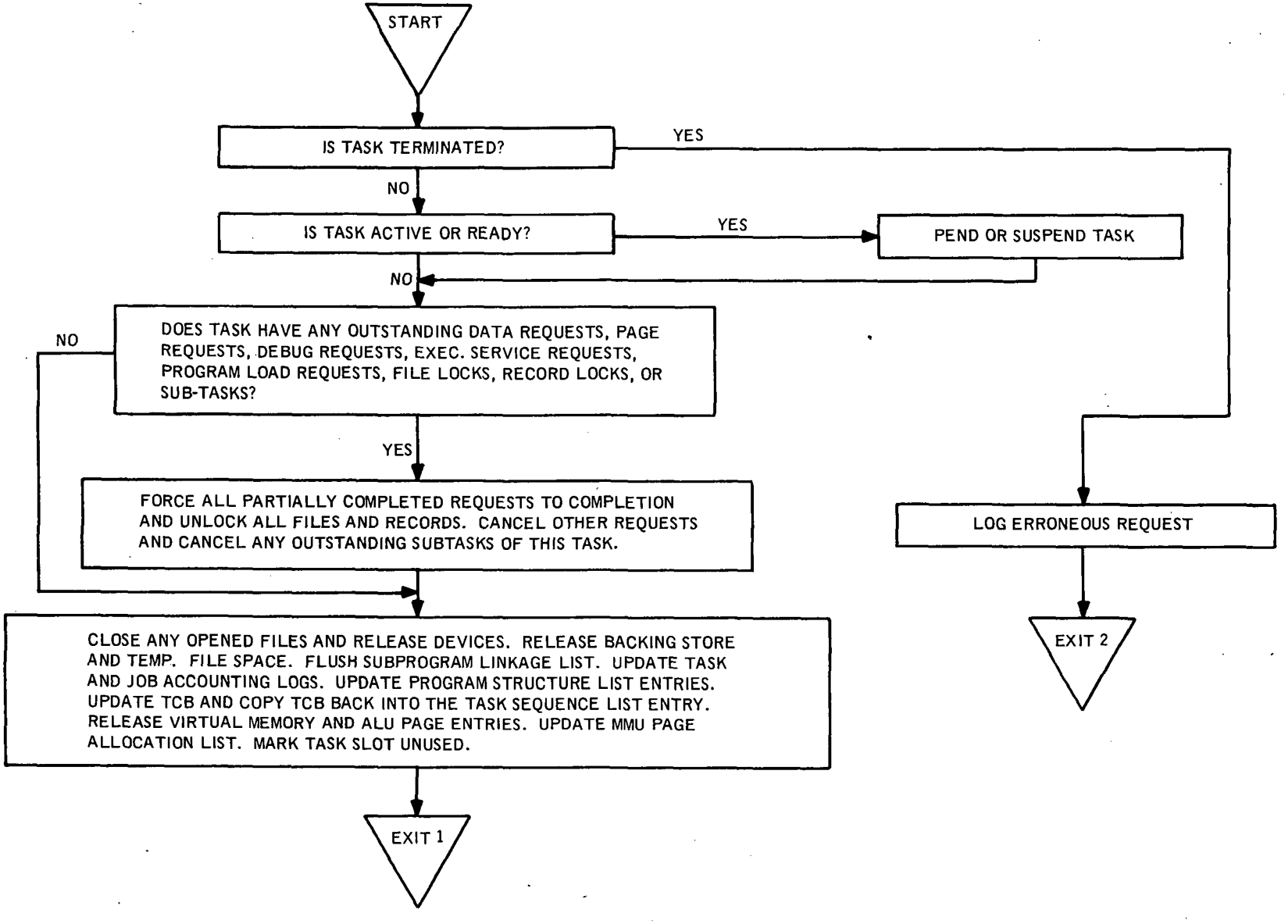


Fig. 4-3. Task termination.

termination the task's TCB is marked to show whether it was Terminated or Cancelled and whether a normal task initiation can be attempted at some future time.

A Terminated task has no resources allocated to it as a task, although the job it belongs to may have resources assigned for the duration of job execution.

4.2 IOU SUBCHANNEL SERVICING

The IOU subchannel routines use a special microcode sequence shown in Fig. 4-4 for servicing the transfer of data between the IOU and the Main Data Bus. This sequence will also manage the transfer of data between the MMUs and the IOU. The rate of executing sequences must be balanced to the data transfer capability of the peripheral equipment. If, for example, a disc transfer were assigned to subchannel 6 and a tape station were assigned to subchannel 1 of the same IOU, then the microcode sequence to service data transfers over the data bus should be executed more often for subchannel 6 than for subchannel 1.

The function of the IOU subchannel Periodic Calling Routine is to initiate execution of microcode sequences for the subchannels at the appropriate rate. This routine will in turn be activated by an internal timer. Upon checking to see if any subchannel's sequence is to be run and finding one, the routine will try to gain control of the bus by using the LGO instruction. If the bus is available, the subsequent portion of the Periodic Calling Routine will be unlocked.

Which subchannels should be activated at any particular time is designated by a mask table which is shown in Fig. 4-5. When the calling routine is run, it will step through the entries in the table and activate microcode sequences for each subchannel indicated. When more than one subchannel microcode sequence is to be activated at one step, priority level determines the order in which each subchannel is serviced. The Periodic Calling Routine waits for completion of the run of all the activated subchannel sequences. When the calling routine resumes, it relinquishes the bus and then suspends itself until the next call from the timer.

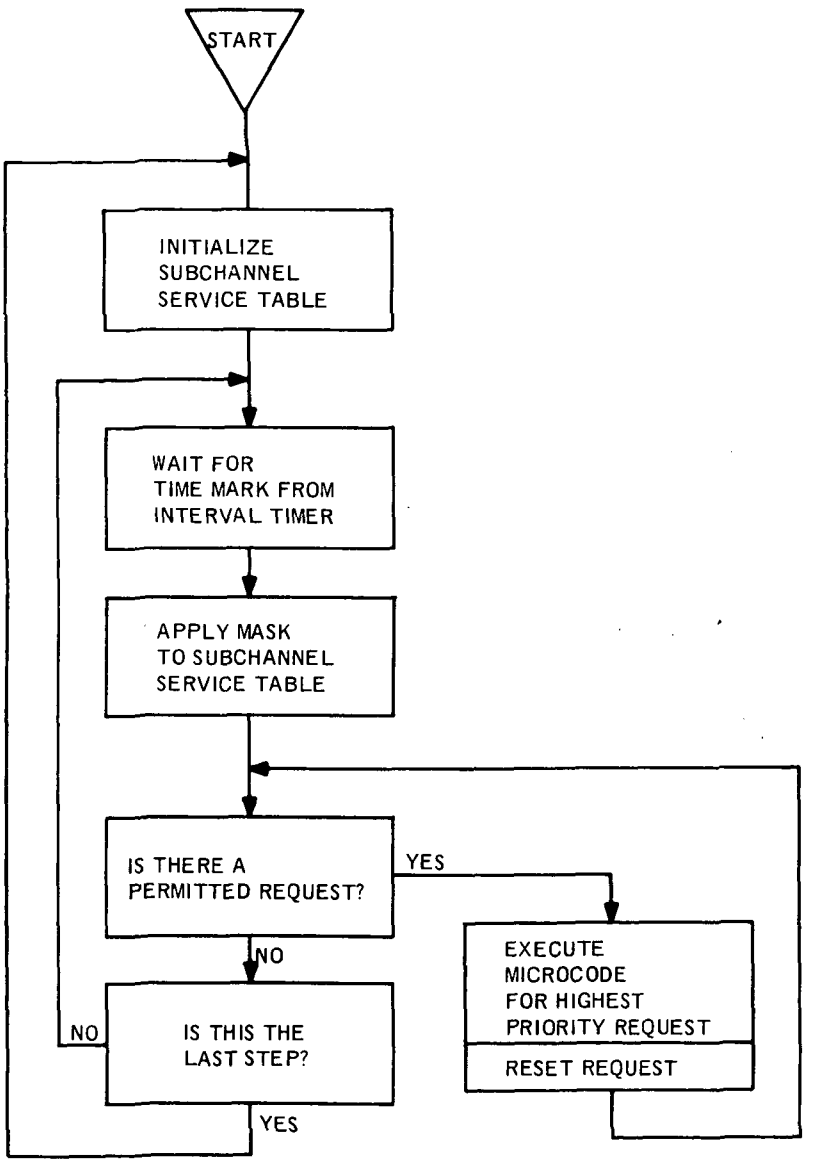


Fig. 4-4. Subchannel servicing.

4.3 PAGING OPERATIONS

In translating virtual to physical addresses, an ATU's CAM may find no stored page address translation. The Virtual Memory Allocation lists in main memory must be searched with one of three outcomes:

- Page is in an MMU
- Page is in the auxiliary memory
- Page is not in the system memories.

The last outcome indicates an error in programming I/O from peripheral devices; and completion of the task, for which the physical address was sought, is impossible without some programming correction. The other two outcomes will result in a valid translation. If the required page is in main memory, reference to it must be loaded into the CAM. If the page is in auxiliary memory, but not in main memory, it must be copied into main memory.

A paging operation occurs in the latter situation. The Page-In-Page-Out procedure locates the least-recently-used page in an assigned MMU, or from within a group of assigned MMUs, and marks the page for replacement. Examination of a control word in this page indicates whether the page can be safely overlaid; i.e., it has not been changed since last copied into main memory, and the parent copy is at an appropriate level of availability. If the page can be overlaid, an I/O transfer between auxiliary memory and the involved MMU is set up to read-in words at the maximum rate of access to the auxiliary memory. If the page cannot be overlaid, an I/O transfer between auxiliary memory locations and an exchange of words in the MMU page frame must be set up. Because the overlay procedure is not only simpler but faster, it will always be used when it will not destroy data.

Whenever an ATU detects a page fault, its CPU is interrupted to task level 8 which will put the CPU under control of the Page Fault Routine. Significant system activity and decisions included in this routine are shown in Fig. 4-6. The CPU must request

STEP NO.	SUBCHANNEL						
	7	6	5	4	3	2	1
1	X	X					
2						X	X
3	X		X				
4		X			X		
5	X			X			
6			X				
7	X	X					
N			X	X			

Fig. 4-5. Subchannel sequence activation mask table.

service from some IOU by setting a bit in its CPU-to-IOU mailbox (see Sec. 4.4 following) and setting up a search for reply after fixed interval.

An IOU takes up the routine upon detecting the CPU request in the mailbox, interrupts to task level 10, and searches the MMU Page Allocation Lists to find the page. If the page must be retrieved from auxiliary memory, the Page-In-Page-Out procedure is called. Since the transfer of words is intermittent, each transfer is upon interrupt only at task level 10. Between transfers, the IOU can process at lower priority levels.

At the end of a page transmission the IOU must update the MMU Page Allocation and Virtual Memory Allocation Lists and leave a message in the IOU-to-CPU mailbox for the waiting CPU, which, upon detecting the message, goes to task level 11 to supply the appropriate reference to the CAM in its ATU. The ATU then can make direct references to the retrieved page.

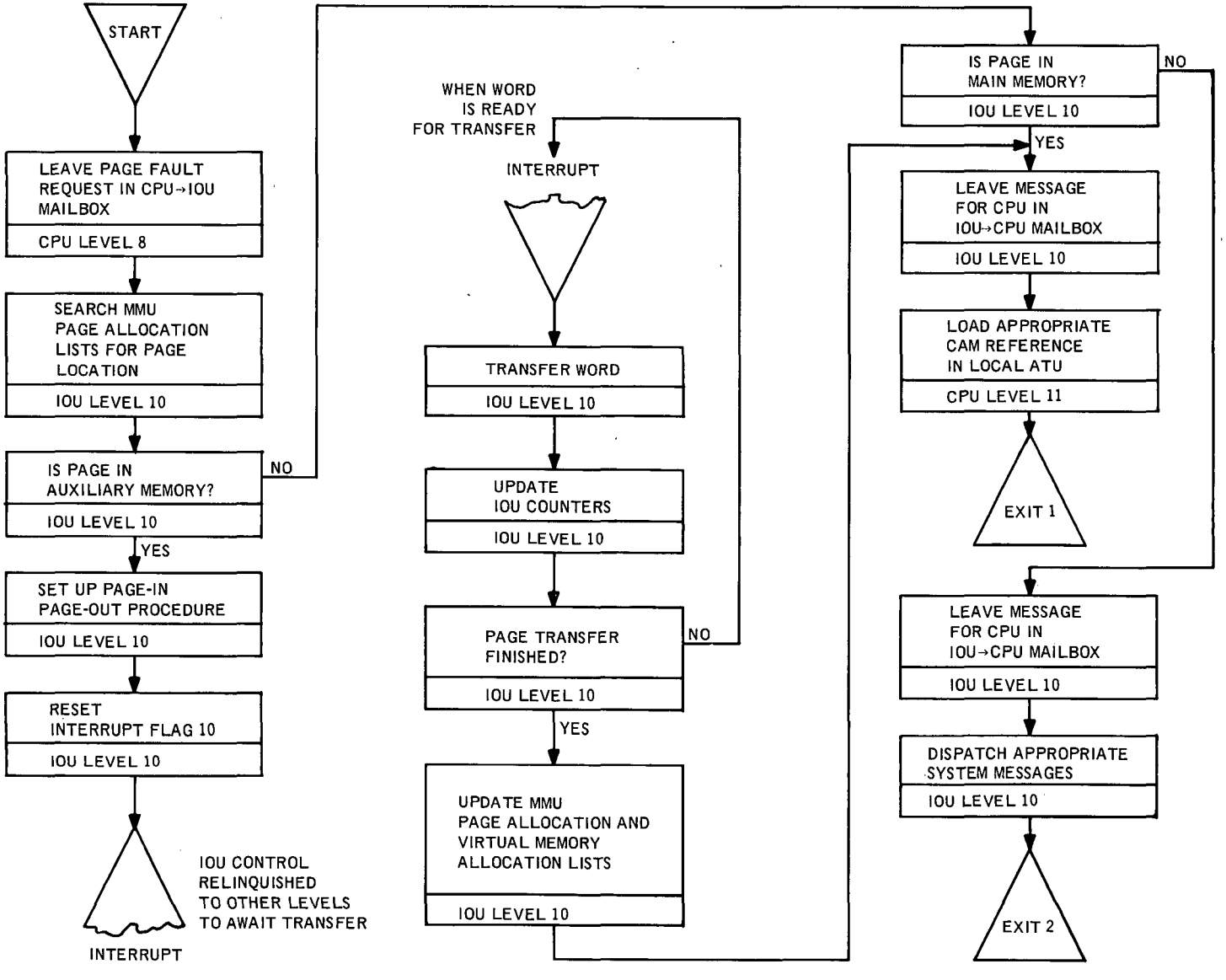


Fig. 4-6. Page fault routine.

4.4 INTERPROCESSOR COMMUNICATION

Several interconnection mechanisms could be used to implement data and control signal transmission between CPUs and IOUs in a SUMC multiprocessor configuration. These include crossbar switches, direct exchange channels, and conventional data buses.

A desire to minimize system complexity and failure modes and a realization that most information exchanged between CPUs and IOUs already resided in a common MMU led to selection of an alternative. Additional MMU monitoring by the IOU was what was needed to complete the processor-to-processor communications. The complete facility is called the MMU "mailbox."

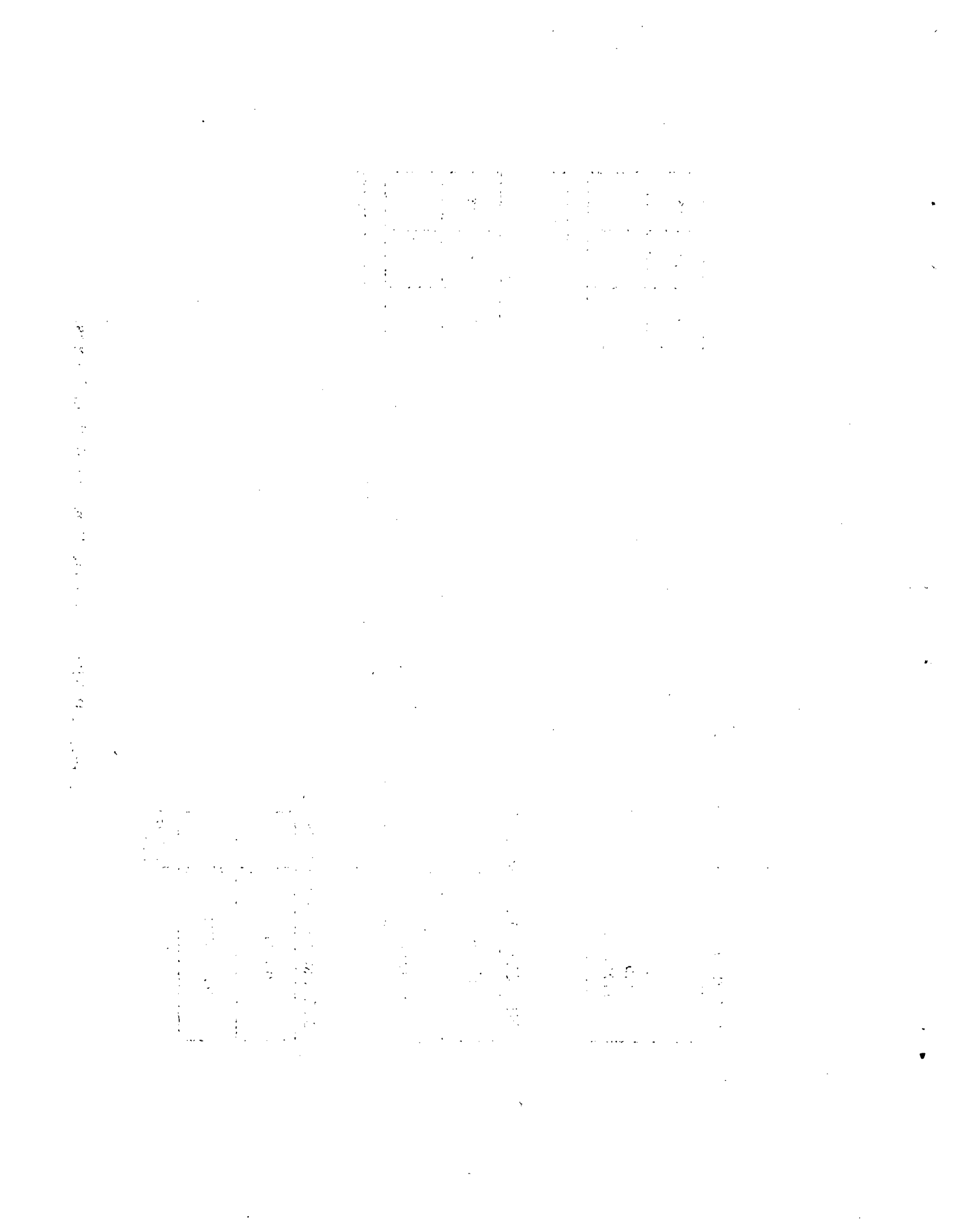
4.4.1 Hardware/Firmware Support of the Mailbox Facility

The following facilities are employed to implement the mailbox:

- Lock-and-Go instruction to preclude destructive interference between processors. This new instruction is included in both the CPU and IOU instruction sets.
- A Multibyte Zero Test instruction to allow IOUs an expeditious check of the mailbox locations in the MMU for service requests placed there by CPUs and to allow CPUs an expeditious check of the status of requests previously issued to the IOUs. The Multibyte Zero Test is similar to Translate and Test, a standard IBM S/360 instruction.
- A timer-driven mailbox polling routine implemented in microcode in each IOU, guaranteeing that no CPU service request will remain outstanding without a response from an IOU longer than a specified interval. This polling routine is executed under control of an IOU system task running at a high-priority interrupt level.

4.4.2 Mailbox Configuration

Mailboxes shown in Fig. 4-7 are of three varieties: the CPU-to-IOU type, holding request indicators; and the IOU-to-CPU type, holding status indicators. For each CPU in the multiprocessor system, there is a request mailbox of the CPU-to-IOU



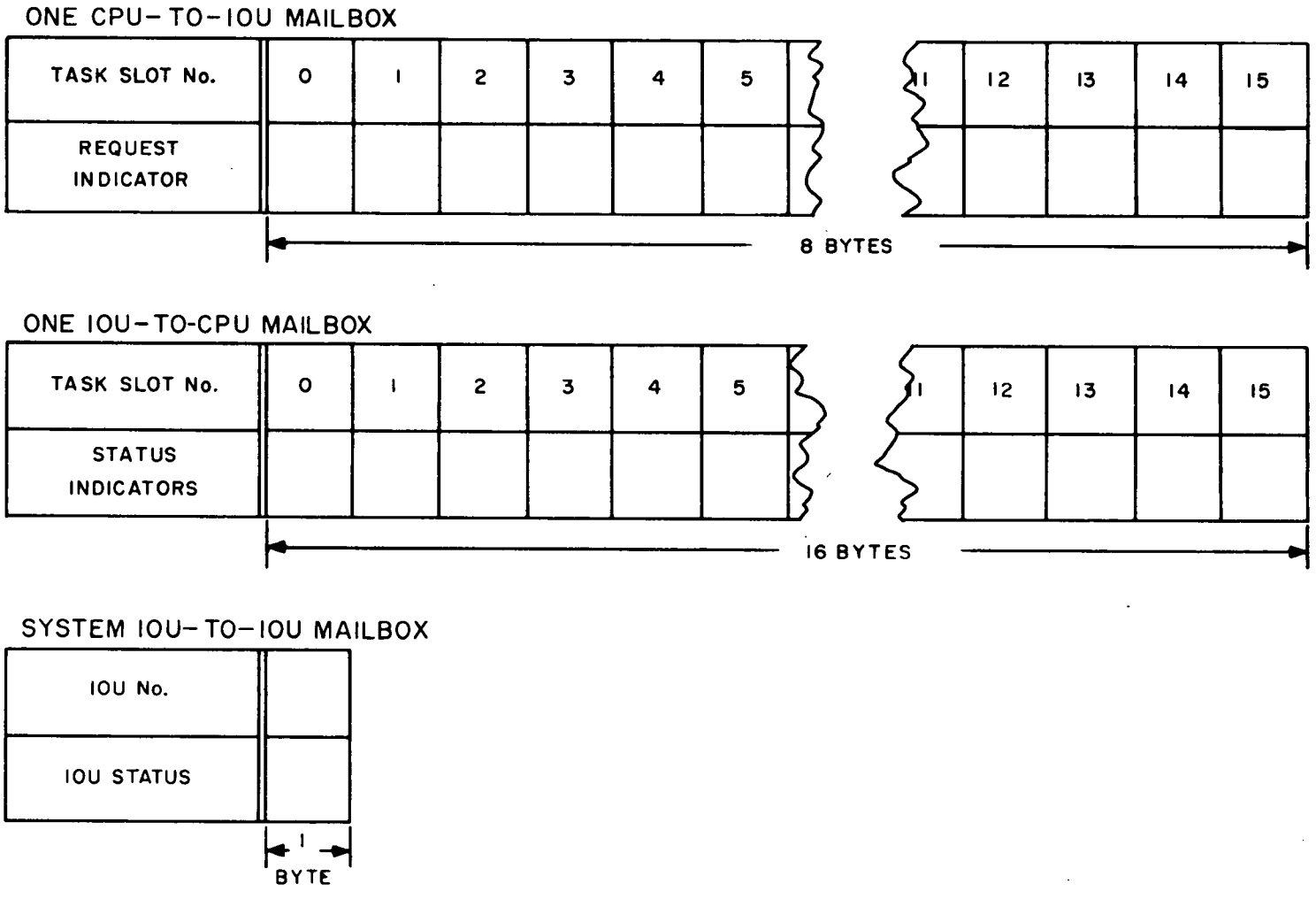


Fig. 4-7. CPU/IOU communications via MMU mailboxes.

type, each of which is implemented by eight contiguous bytes in main memory and each of which is subdivided into eight equal partitions, one for each Task Slot. For each CPU there is a status mailbox of the IOU-to-CPU type which is implemented in 16 contiguous bytes of main memory, and likewise each partition (byte-sized in this mailbox) will be dedicated to each of the eight CPU Task Slots. A single system mailbox will be provided for IOU-to-IOU communication.

The request indicators, a separate bit for each, include those for:

- Power failure
- Page fault
- Data request
- Module failure.

The status indicators chiefly signal that an I/O task has been completed, but their interpretation depends upon the type of original request.

4.5 SYSTEM STATE TRANSITION CONTROL

Individual unit state control is necessary to guarantee the orderly progress of each unit from power-on to operating and back to power-down without disturbing other units in the system. Important features in the design of any state control system for multiprocessor are:

- A set of defined states and transitions
- The ability of an individual unit to execute its own executive functions as directed by the system executive code
- The ability to have like units simultaneously in different states.

Multiplicity of CPUs and IOUs in a system make a variety of combinations of states possible; however, only the few that will favor a high level of computational throughput will be allowed.

4.5.1 System States

The MMUs will be limited to two states: "On" or "Off". While On, use of an MMU will be under assignment by the system executive. Access to an MMU is controlled jointly by logic of the ATU and the MMU itself. While Off, an MMU will be in the power-down state. Switching an MMU On or Off will be by manual control at the operator's console.

When an MMU is switched On or switched Off, the system executive will be notified, and such notification will be automatically relayed to the operator.

The CPUs and IOUs require seven operating states:

- Power-down
- Initial
- Isolated
- Idle
- Self Test
- Multiprocessor computing
- Triple modular redundant computing.

Most of the transitions will be initiated by command message from the operator's console. One exception is transition from the Power-down state to the Initial state. This transition must be directly commanded by the operator from the console. Almost all of these transitions are single-step, with two exceptions: the two transitions between the two computing modes -- Multiprocessing and TMR.

The state diagram is shown in Fig. 4-8. There is no simplex computing state although when three processors operate in the TMR state, the fourth unit would have to be operating in the simplex. It is expected that the fourth unit would be using the multiprocessor executive so that it will be in the multiprocessor computing mode. Transition to and from the TMR state must be via the Multiprocessor state. Since

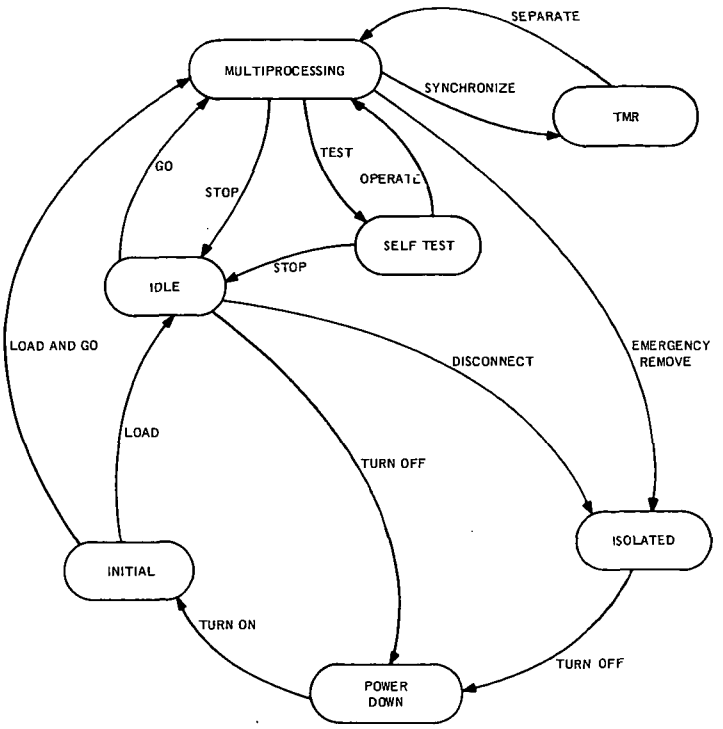


Fig. 4-8. Processor state diagram.

TMR computing must be done with at least three operational units, any joining or disconnecting of processor service must start or end at the Multiprocessor state.

The transitions into and out of TMR computing are the only multi-step transitions. The manners of transition are shown in Fig. 4-9.

4.5.2 Initial Turn-On and Loading

The state of a computer immediately following the application of power must be anticipated. Registers have either an unknown content or, with the inclusion of a power-on master-clear, have specified contents. The same is true of semiconductor memory elements, with the exception of read-only-memory. Thus it is advantageous to store basic firmware and initial program load routines in ROMs.

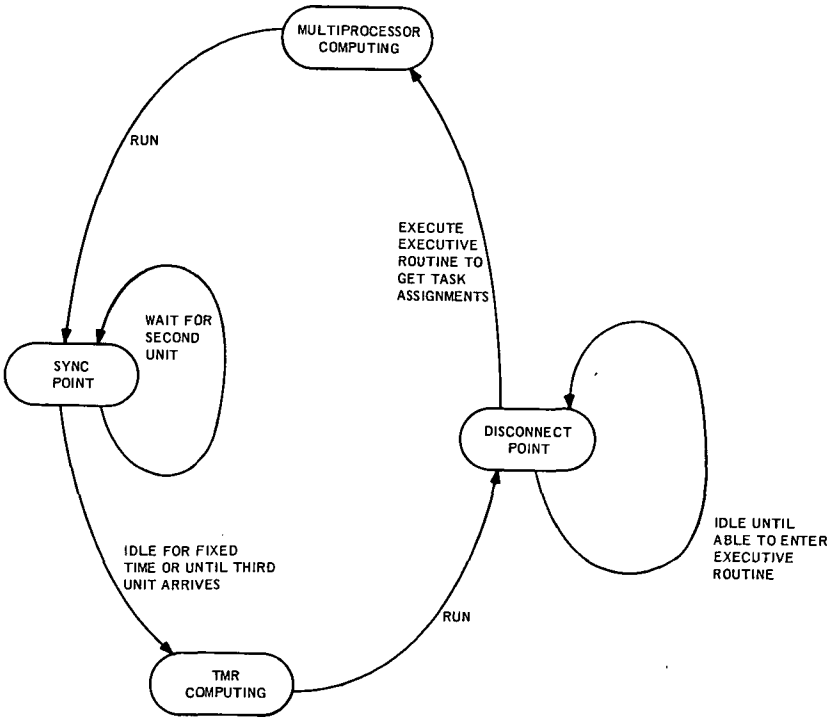


Fig. 4-9. Detailed computing state transitions.

Initially, there is no routine in a main memory for a processor to execute and no entry in the ATU with which the processor could access the program code even if there were a routine. The contents of the ATU tables could be loaded automatically if they were in memory and if an interrupt could be produced, but no interrupt could normally be expected, or made, to occur. Even if an interrupt were commanded, there would be no PSW and no task level indicated as destination in the event of an interrupt.

Any CPU/IOU that is initialized must first be in a sufficiently determinate state to enable certain essential activities such as writing-in a PSW and loading ATU tables. Then the CPU/IOU should allow initial program load and bootstrapping to load the operating system and task programs into main memory via the data bus from peripheral devices. This loading must be performed by an IOU which will upon power-on be initialized to run a firmware sequence which will run with inhibited instruction fetch and test for interrupt until the initial load operations are complete.

The firmware sequence for loading begins register-clear operations as needed. A poll (low rate if desired) of the interactive console starts and continues until

answered. Then the IOU fetches the following information from the console in order to enable the next step: the PSW contents for one subchannel, ATU table entries to allow sequence data to be stored, and a description of the input operation to load the memory. These data could be sent sequentially, even one byte at a time, if sent directly from the console. If the console is capable of off-line data composition, this information could be composed before the poll is answered. Pressing the Load, or Load-and-Go, button on the console permits the polling to be answered.

The input operation specified by the initial console action could be limited to that of loading a bootstrap routine, after which the load control would be transferred to the bootstrap routine. Both the bootstrap routine and the program it loads allow entries in the ATU tables.

Since there is no direct connection between a CPU and the main data bus, a different procedure gets a CPU operating. This procedure must use the only means of communication that the CPU has -- message areas of main memory. Upon power-on, the CPU using a special firmware routine loads the minimum complement of registers to bootstrap into an operating state. The memory data loading operation must contain no instruction fetch and no recognize interrupts.

The chief difference in the CPU initial load and an IOU initial load is that the CPU presumes initial loading of main memory by an IOU. The CPU must detect when the IOU has completed the main memory load operation before it attempts to use main memory data to begin its own load. Therefore, the last word that the IOU stores in a dedicated location should be the first memory word that the CPU will receive.

There are two possible ways that a CPU can get this initial data from main memory. Data from read-only control memory could create a physical memory address and request a memory read operation. When the CPU receives a significant word from memory, it performs several memory read operations until it has fetched and stored in its PSW and ATU registers the data it needs for a bootstrap operation.

The problem with this approach is that the firmware generates a physical address for main memory rather than a relocatable address. The main memory unit addressed must be assumed operational.

An alternate approach would be to have the CPU start by continually reading data from main memory. The IOU, upon loading the main memory with its last word, would allow transfer of the word to the CPU. Upon receiving it, the CPU would use it as the main memory entry to the data it needs for initialization.

This approach requires further capability in the system. If the memory performs the sending operation, it must be able to do so without having been first requested to by the intended receiver. Complementary capability must be in the IOU in order to command this action. It would be possible for the IOU to send the data directly if both the CPU and the IOU had an interconnecting bus. Were a frequency multiplexing scheme used on the Internal Data Bus, the direct transfer could be effected by having the IOU transmit on the same carrier frequency on which the CPU is receiving. If all CPUs in the initialization process receive on a common channel, one data transfer from an IOU would be sufficient.

To give the system operator full control, CPU initialization would be contingent upon pressing the CPU Start button on the console.

4.5.3 Configuration Control by Operator

Like any other interactive operation, operator control of the system configuration can be from a remote console. With the presence of communication supporting software, the operator may initiate a message that will specify the state of any CPU, IOU, or MMU. To optimize state control operations some control messages must be generated upon setting individual control switches on the console. Other control messages could be composed by the operator using the off-line composition feature and then sent to the system.

A configuration control message contains the identity of sender and of receiving unit and instructions for the receiving unit. Including the sending unit ID allows analysis of authorization of the sending unit for the type of message. The sender's ID part of the message format being used, a reconfiguration message would be written into the system via the console, without any display of the entered contents. The ID for the receiving unit will provide for multiple unit addressing. This feature will be of value when a change from multiprocessing state to redundant state is being requested. A communication support routine will check the action being requested in order to ensure proper resource allocation. If some minimum resource limit must be exceeded, the console operator would be requested to confirm the original message before it is acted upon. Since there is a limited set of actions that can be performed, the coding for the requested action will be satisfied by a limited vocabulary.

4.5.4 System Configuration Control

Reconfiguration results from a routine to be implemented at a particular time as one of the tasks of the system. Reconfiguration may require operator approval before execution. Messages will display a description of the projected action, and operator approvals are communicated to the system by having the operator place his identity code in the sender's ID field and then dispatch the message. Modifications to the planned action could be inserted by the operator before sending. One modification might be to select an alternate unit as receiver.

4.6 FAILURE RESPONSE

So far two recipients of the fastest response of the multilevel interrupt for execution of a specific program with minimum of delay have been identified. The highest priority level routine will be run in event of power failure. The next level routine will be run in event of a processing unit scratchpad failure.

4.6.1 Power Failure

Each processor will have its own power supply, with individual power failure detection system. Upon detection of a power failure, an interrupt at the highest priority level will occur. The routine run at this level will supervise an orderly shut down of the failing processor and will require main memory storage of the current contents of all internal registers: PSW Registers, General Registers, and Floating Point Registers. Contents of other processor and ATU registers are not stored.

Having stored its register contents in memory, the processor posts the power failure code in the CPU-to-IOU mailbox or system IOU-to-IOU mailbox as appropriate, so that the executive routine executed by another processor will be able to detect the failure. Actions such as operator notification, transfer of the stored register content information to the auxiliary memory, and transfer to auxiliary memory of main memory contents associated with the failed processor's tasks, request the services of any IOU that is free to act upon the failure notice.

Finally, the failing processor will put itself into the Idle state so that when the power goes off, the processor will not cause a transient that will disturb the system. By going to the Idle state instead of to the Isolated state, the processor may be re-activated if power is restored before the processor is completely shut down.

4.6.2 Scratchpad Memory Error

Error detection capability included in the processor scratchpad memory will initiate a Scratchpad Error Routine (see flowchart, Fig. 4-10). For elementary operations (EOs) that fetch data from the scratchpad memory, the retry control will cause the EO to be repeated up to four times whenever an error is detected in the SPM readout.

Anticipating the repetition, retry control inhibits use of the result of any EO operation in which the readout error is detected; e.g., if the EO is performing an RR Add, the use of the adder output is inhibited upon detection of an SPM readout error. Any repeatable instruction may be retried.

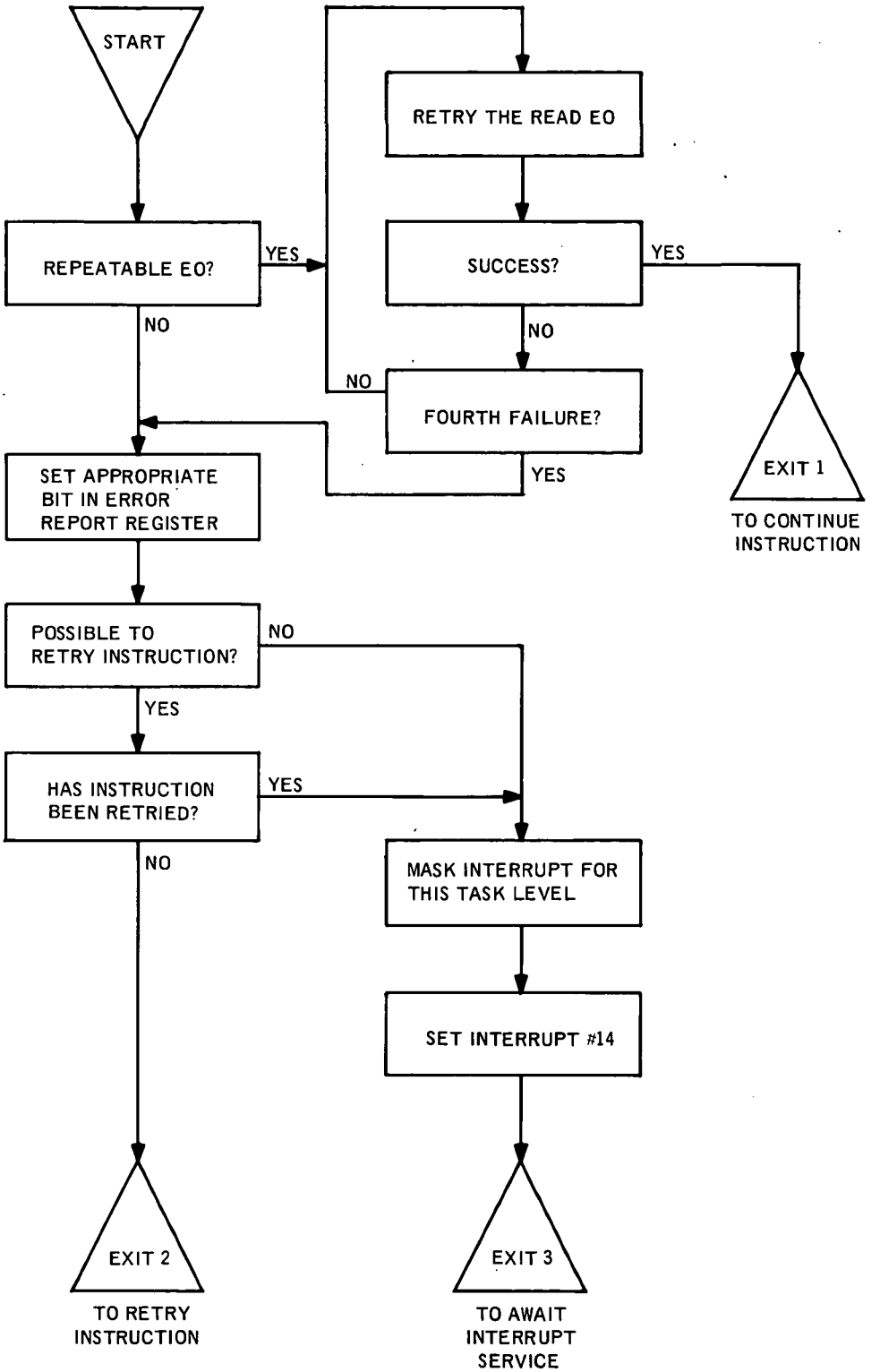


Fig. 4-10. Scratchpad memory error routine.

If the readout is still faulty, a change of control to task level 14 is then performed. The routine executed at priority level 14 will be a diagnostic and recovery sequence.

The following recovery procedure is proposed (see Fig. 4-11).

1) Determine which priority level was active when the error occurred.
2) Test the contents of all scratchpad memory locations to find the error.
3) Correct, if possible, the stored word containing the error and return control to the previous instruction.
4) If the memory location is good but the faulty data cannot be corrected, send the proper notice to the Executive and call the Task Assignment Executive Routine.
5) If the memory location itself is bad, notify the Executive that the task level that was active when the error occurred cannot be used. Then call the Task Assignment Executive Routine.
6) If the Task Assignment Executive Routine is not operational at its original priority level, the processor sends an SPM failure message to the Executive and enters the Idle state.

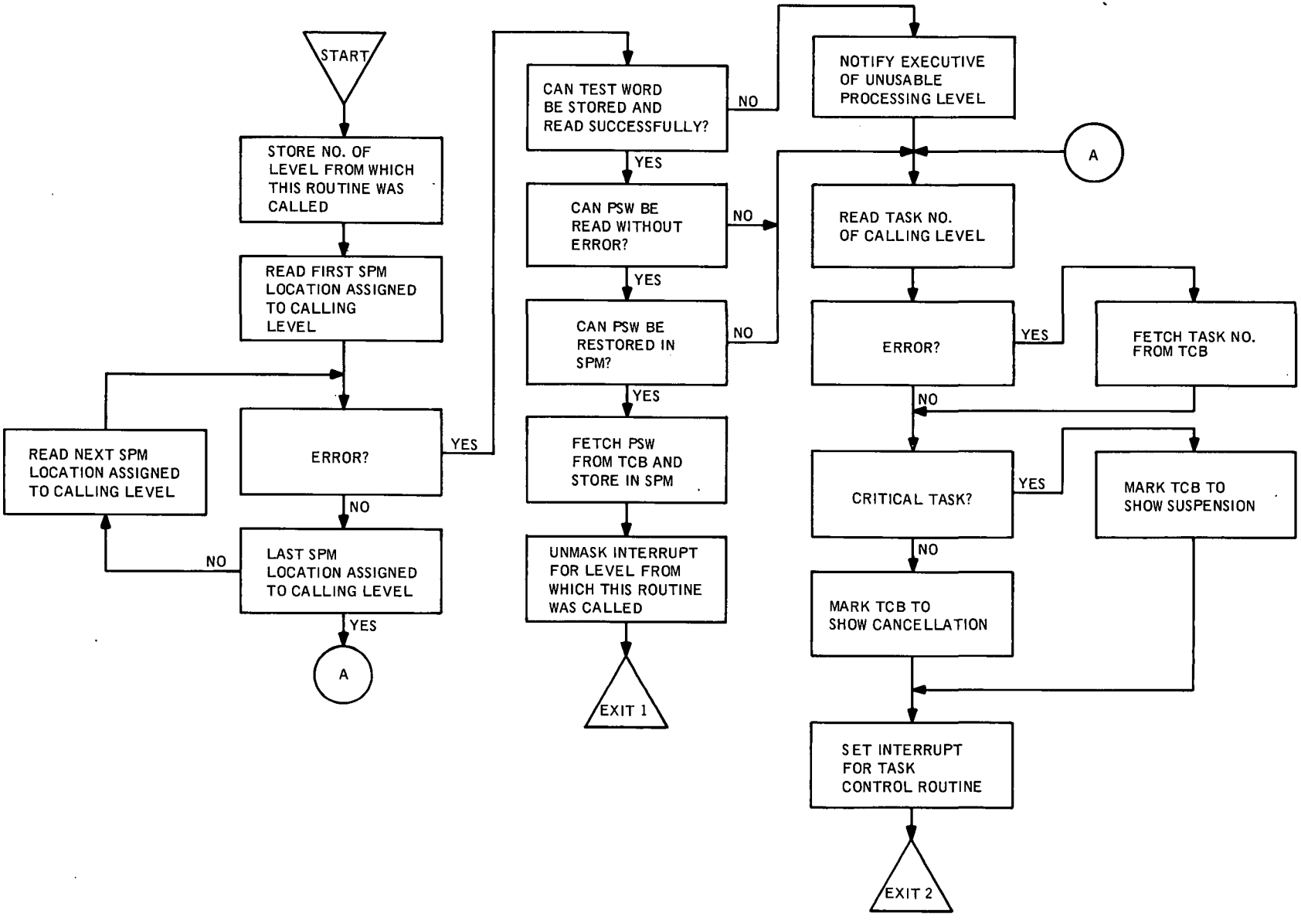


Fig. 4-11. Level 14 procedure for scratchpad error.

Section 5.0

EXTENDED STUDIES

5.1 INTRODUCTION

No matter how extensive, no matter how well planned and executed, a study would not have been imaginatively pursued if it did not reveal a multitude of details of interest and profit for further study. The potentialities of the ideas advanced in the previous text have in part been suggested by intuition and in part by preliminary calculation. Corroborative studies should be undertaken to establish these details and to explore their further potentialities before hardware and software dollars become totally committed.

There is no question that multiple sets of hardware can be interconnected in a fashion to provide simultaneous operation. The real question is how effective the operation of the particular collection of hardware with selected interconnection techniques is in providing computing power. This section attempts to define fruitful areas of further study effort in order to evaluate more completely the effectiveness of the multiprocessor system proposed in this report. It is hoped that the following observations and suggestions will enhance the value of the present study as one member of a series of systems studies for the SUMC and CVT. To realize full benefits from the multiprocessor system and software work of IBM, Intermetrics, CSC, and now from RCA, the following suggestions for continuing study are made.

Four principal areas have been identified as convenient nuclei for proposed study topics:

- Coordination and timing analysis
- Traffic accommodation analysis
- Mechanizations studies
- Reliability studies.

It is recognized that a certain overlap in topical scope is natural and could be beneficial if extended studies had to be partitioned for expedience and then synthesized for final overall results. Software might merit a topic of its own, but in its functional aspects it is so intimately a part of the system that to separate it would be a step backward into hard-and-fast software and hardware compartments, which are not productive from a systems point of view.

5.2 SYSTEM COORDINATION AND TIMING ANALYSIS

5.2.1 Synchronous and Asynchronous Operation

Coordination and timing in a multiprocessor is the key to balancing unit employment and thereby keeping up overall throughput rates. Spatially distributed autonomous units, efficient utilization of time, and real-time responsiveness dictate a large amount of asynchronous operation. To this end a simple, yet powerful interrupt system has been proposed for CPU/IOUs. Simplicity and order, however, are characteristic of synchronous operation. Within individual units, elementary operations will be governed by regular periodicity. The questions here raised for more rigorous analysis and detailed design are:

- What is the ideal extent of asynchronous and of synchronous control in order to obtain the most advantageous mixture of the two techniques?
- How is reliable and uniform distribution of clocking signals obtained for the synchronous control?
- How much signalling and response (handshaking) constitute a minimum necessary for maintaining required levels of reliability?

Polling as proposed for data exchange between the MMUs, and the bulk store peripheral units occupies a gray but significant area where the synchronous and the asynchronous meet. An IOU polls on a periodic basis -- albeit the unit may be carrying out pollings of different periods simultaneously. The vehicle of the polling, the main data bus, however, may under emergency configuration see polling from several IOUs asynchronously. This situation is handled in the proposed system by software lockout (see LGO instruction). The adequacy of this scheme must be analyzed from the point of view

of timing, deadlock proofing, and reliability. Analysis of adequacy will depend on projected demands upon the main data bus; and until specifications are written for I/O traffic, equations of the analysis can only be set up and solved for sensitivity factors.

5.2.2 Compatibility of Activity Rates

In order not to induce "binds" during processing, rates of data handling must be matched. Part of this problem is in the rates of information transfer for each module in the system. In the proposed system, the transfer rate of MMUs is ten times that of auxiliary memory. Such a figure awaits confirmation on the basis of calculation or simulation and in conjunction with pertinent rates for the rest of the system.

The very integrity of a multiprocessor makes this an overall timing operation, with each reworking of the timing increasing the system efficiency. Size, complexity, and variability of the components dictate that insight and care must be used to ensure convergence of the timing process. Because of the potentially high throughput, however, the rewards of correct timing are great both as to total jobs handled and as to decreased required buffering. Physically this could mean less auxiliary memory and could realize a saving of weight and space.

For a system as powerful as the SUMC multiprocessor, timing after construction is extravagant. The changes dictated might be wasteful or prohibitively expensive, and the time to do it nonexistent. During design, however, the system can be tuned if a simulation model has been concurrently built. System modeling and simulation are to be recommended for systems of this size and complexity.

The recommended simulation of data exchange rates and processing rates deals with tried-and-true procedures which would need no simulation in themselves but would require one because of their interrelationships.

Use of mailboxes for interunit communication, however, is not standard operating procedure. Implementation of mailbox message handling must be worked out and timed so that its competitiveness with a more conventional mode -- the use of an interprocessor bus -- may be calculated. The comparison is not simple because use of an interprocessor bus in a multiprocessor can mean direct transmission to multiple receivers with the first-ready receiver the responder and all others notified of cancellation of message. Counterpart of this action for mailbox transmission must involve firmware and some software in the form of Lock-and-Go so as to award the response task to only one, but any one, unit.

5.3 TRAFFIC ACCOMMODATION ANALYSIS

5.3.1 System Simulation as a Design Tool

The system simulation is an ideal tool for the study of traffic problems and for experimentation with their relief. System simulation could answer questions as to sufficiency of parallel paths and alternate facilities and as to adequacy of storage and buffering. These are the static questions of the first order, and they are vital to sizing of the system. Because complex queuing situations are involved, they are intractable by hand calculation.

Of even greater interest and greater subtlety are the second order questions: those which are peculiar to dynamic situations. These are scheduling problems vital to the stability of the system in action. The problem of multiprocessor task scheduling has stimulated research, but the insight so gained requires application and testing in particular multiprocessing situations. * Correct solutions will assure a viable operating system which will commutate tasks so as to maximize throughput and maintain an even flow of data into and out of the main memories.

*Muntz, Ref. 17.

An illustration will be taken by citing the paging example in Sec. 2.5.3.2. This presimulation speculation in a dynamic area shows how throughput might be sensitive to an interfault number (the number of consecutive accesses to main memory without requiring retrieval of a new page). If, for example, it were critical to hold fluctuation of throughput rate to a narrow band despite a wide variation of interfault number, then something would have to be done to system parameters in order to decrease the slope of the sensitivity curve, i.e., desensitize the system to variation of interfault number. With a simple algebraic expression the alternatives are clear, but seldom can the real situation be described in simple algebraic terms. It is here that a faithful system model must be constructed and exercised in a "realistic" environment with "realistic" inputs. Study of different dynamic profiles, which can be collected in a single simulation run, will give insight as to how to desensitize the system. A few cut-and-try simulation runs with the model will prove it. Results of the simulation can be passed on to system designers as guidelines for construction or modification. The same results will be of interest to software designers who may participate in relieving problems by tuning the scheduling algorithm to tolerate wider throughput rate fluctuations or by modifying the compilation and memory paging schemes so as to increase the interfault number.

Interfault sensitivity is but one of many sensitivities that might well be studied profitably by simulation. Experience in the dynamics of the system will motivate speculation as to behavior, adequacy, and efficiency of the system into a third order realm, entry to which may not be necessary if simulations of current designs prove their behavior satisfactory.

In passing it should be noted that a significant advantage of system simulation is an almost mechanical one. When a model is being built concurrently with a design, the designer is forced to make his ideas explicit and understandable to the modeller. A byproduct of the transfer is the elimination of a number of inconsistencies and incompatibilities. Thus, modeling serves as a meticulous design review.

5.3.2 System Deadlock Potentiality

The great multiprocessor problem has not been so much in building multiple quantities of miniaturized hardware or even in distributing signals, although problems of delay, skew, crosstalk, and signal degradation abound; it has been in whether the multiprocessor can handle a wide variety of traffic situations without deadlock and without idling large amounts of hardware. Cures for deadlock have been suggested (Sec. 2.3.6) for the proposed multiprocessor. How effective these cures really are can be measured only from a system simulation or from the constructed system. As mentioned before, by the time the system is on the floor, it is too late.

5.3.3 Main Memory Traffic

As has been mentioned, paging is expected to constitute a significant portion of traffic to and from MMUs. If the proposed rates of transfer and cycling are maintained, an MMU, while paging, will be at least 10 percent access loaded. Such loading may appear small but it is of high priority and becomes an ever greater threat to the dynamic stability of memory unit operation if access-loading reaches a moderate (75 percent) size.

When access to an MMU is loaded to 30 percent, i.e., 30 out of 100 possible accesses averaged over a relatively long period, short term queues for access are possible, even with a random distributuon of interaccess times. Buffering and waiting will be necessary. A study must specify a reasonable amount of hardware and for buffers and a reasonable amount of throughput slack to accommodate peaks.

A small but important item of MMU traffic is the number and timing of LGO instructions. It is obvious that LGO can be an effective lock, but how fast it can operate to turn aside other requests for the protected code while enabling the authorized task to use the code with the immediacy needed is in question. Whether several LGOs to different areas of protected code in a single MMU can be tolerated without

deleterious effect on throughput must be established before an operating system can depend upon it.

Similar studies of feasibility, i. e. , not whether the proposed schemes will work but whether they will work well enough, must be made regarding mailbox communication. How much access-loading mailbox messages represent is a function of both user and executive requirements in communication. Total amount of traffic will not, however, be as critical as how speedily messages get through. Because messages must be "picked up" either on timed stimulus or when a processing unit is idle rather than upon their generation, effectiveness of the proposed plan must be demonstrated in either a well detailed or a well parameterized simulation.

5. 3. 4 Executive System Traffic

Two features of the selected multiprocessing system engender executive system activity that involve MMUs: the interrupt system and memory management. Both feature the accessing and updating of master and derivative lists. It is a matter of speculation that the housekeeping can be done without degrading throughput too much.

Within the interrupt handling software or firmware, it may, for overall throughput, be profitable to adjust task priority dynamically. It is easy to invent a number of algorithms which will favor task parameters such as: length of task, numbers and types of system resources required, time since last activation contributing to the job or task, and a running evaluation of the importance of tasks to the entire work of the system. The results of a simulation from a model of the interrupt system, which would include interaction and competition (via LGO) of several processing units, would favor a selection of algorithms and would aid in identification of the parameters of the situation in which each algorithm was the most advantageous.

5.4 MECHANIZATION STUDIES

Much is known and has already been proved concerning the design, construction, and microprogramming of individual units. The particular requirements of the SUMC multiprocessor that call for unusual study in mechanization are: the means of interconnection for data and the error checking logic.

5.4.1 Bus Interconnections

The interconnection problem has been brought out in Sec. 2.2 but detailed attention should be given to development and testing of the proposed FM multiplexing. Considerations of the bus interconnection involve a balancing of numbers of parallel physical channels v. numbers of frequency channels and, in the area of modems, complexity v. numbers. The critical size, reliability, speed, and cost will uncover many points of tradeoff which must be established in sequence before the design is achieved.

Design must then be centered in the Data Interface Unit (DIU) which connects digital units, both system and peripheral, to an FM bus. Before physical details of this unit can be described, specification of the required interfacing (data, control, error check) must be written and the tradeoffs mentioned above must be made.

5.4.2 Error Checking Logic

As will be called for by further study suggested in Sec. 5.5, logic for the checking and generation of error-detect-and-correct code must be provided at points found strategic and necessary to meet criteria of reliability. Physically the logic must add little bulk to the system; but, more importantly, it must not in itself reduce reliability nor should it drag down speeds of data transmission.

5.5 RELIABILITY

As the proposed system emerges, a parallel study aimed at applying theoretical principles of reliability rigorously to the proposed design should be undertaken. As with

the suggested system simulations, such a study should be interactive with system design. Major thrust of this work is seen in two areas: error control and TMR potentiality for increasing reliability.

5.5.1 Error Detection and Control

Basic to any reliability concept is the ability to detect an error. Detection is done for real time applications either by using redundant information (coding) or by redundant hardware (comparison/voting) or by a combination of both. If extra computer time is available prestored diagnostic routines can be executed and compared to prestored results.

After the basic detection of an error, a larger problem occurs — what response is appropriate. Typically error indications anywhere in the computer are ORed in order to generate an interrupt, and the software directs subsequent action. Owing to the complexity of modern computers, real time equipment cannot afford time to process an error detection - error correction routine. The multilevel interrupt structure used for this multiprocessor system should prove a great asset in eliminating software through use of hardware and focusing the error response routines on the particular part of the system in which error was detected. Additionally, each error response can readily be made independent of that part of the processor which was detected as in error.

In order to define a viable error detection and correction technique the multiprocessor system needs additional study in these areas. This study should use a Triple Modular Redundancy (TMR) configuration as a baseline for evaluating tradeoffs of additional complexity v. additional reliability. The TMR configuration has been flown and considerable data on it is available.

5.5.2 TMR Potential for Increasing Reliability

The redundancy inherent in the multiprocessor itself suggests that reliability could be enhanced by legislating redundant behavior and acting on majority results. Triple Modular Redundant (TMR) operation has received some intuitive attention, but again, for serious consideration, theoretical knowledge must be applied to a particular

SUMC multiprocessor system of at least three processing units. The study should establish the tradeoff between gain in reliability and degradation of throughput. Simple intuition that throughput will be reduced to one-third will likely be wrong because comparisons (voting), and competitions for, or common use of, MMUs and other facilities will consume some processing time.

From the standpoint of system design, a number of problems raised solely by TMR operation must be solved:

- How many and where the points of voting must be,
- How the voting is coordinated,
- What to do about the resulting simultaneous requests for commonly used facilities,
- How to deal with voter disagreements, and
- How to coordinate the entrance or exit of any processor to and from this mode of operation.

5.6 CONCLUSION

RCA considers the SUMC Multiprocessor Study here reported as one in a continuum of multiprocessor studies leading to the final design and construction of the multiprocessor systems to which the Astrionics Laboratory of NASA aspires. Consistent with this thought RCA has pointed out these avenues of further study.

REFERENCES

1. E. G. Coffman, Jr., et al., "System deadlocks," Computing Surveys, vol. 3, pp. 67-78, June 1971.

2. E. G. Coffman and L. C. Varian, "Further experimental data on the behavior of programs in a paging environment," Communications of the ACM, vol. 11, pp. 471-474, July 1968.

3. Robert C. Daley and Jack B. Dennis, "Virtual memory, processes, and sharing in MULTICS," Communications of the ACM, vol. 11, pp. 306-312, May 1968.

4. Peter J. Denning, "The working set model for program behavior," Communications of the ACM, vol. 11, pp. 323-333, May 1968.

5. Peter J. Denning, "Thrashing: its causes and prevention," Proc. 1968 FJCC, pp. 915-922.

6. Peter J. Denning, "Virtual memory," Computing Surveys, vol. 2, September 1970.

7. Robert M. Jones, "Factors affecting the efficiency of a virtual memory," IEEE Trans. on Computers, vol. C-18, pp. 1004-1008, November 1969.

8. M. Joseph, "An analysis of paging and program behaviour," The Computer Journal, vol 13, pp. 48-54, February 1970.

9. J. R. Kennedy and W. S. Fitzpatrick, "Spaceborne computer executive routine functional design specification -- volume II: computer executive design for space station/base," final report, contract NAS 8-24930, NASA report CR-1868, Computer Sciences Corp., October 1971.

10. J. R. Kennedy et al., "System configuration and executive requirements specifications for reusable shuttle and space station/base," final report, contract NAS 8-24930, NASA report CR-1820, Computer Sciences Corp., May 1971.

11. Donald E. Knuth, The Art of Computer Programming: Vol. 1. Fundamental Algorithms. Reading, Mass.: Addison-Wesley, 1968.

12. Jaroslav Kral, "One way of estimating frequencies of jumps in a program," Communications of the ACM, vol. 11, pp. 475, July 1968.

13. C. J. Kuehner and B. Randell, "Demand paging in perspective," Proc. 1968 FJCC, vol. 33, pp. 1011-1018.

14. Daniel J. Lasser, "Productivity of multiprogrammed computers -- progress in developing an analytic prediction method," Communications of the ACM, vol. 12, pp. 678-684, December 1969.

15. R. E. McNabb, "Modular space station computer study," IBM report 71W-00345, 27 October 1971.

16. James S. Miller et al., "Multiprocessor computer system study," final report, contract NAS 9-9763, Intermetrics, Inc., March 1970.

17. R. R. Muntz and E. G. Coffman, Jr., "Optimal preemptive scheduling on two-processor systems," IEEE Trans. on Computers, vol. C-18, November 1969.

18. William Wesley Peterson, Error-Correcting Codes. Cambridge: M.I.T. Press, 1961.

19. G. E. Proch, "A conceptual study of the SSV/GN&C system data bus," final report, contract NAS 9-5191, NASA report CR-1792, Lockheed Electronics Co., August 1971.

20. B. Randell, "A note on storage fragmentation and program segmentation," Report RC2102, IBM T. J. Watson Res. Ctr., Yorktown Heights, N. Y., May 1968.

21. P. Wegner, "Machine organization for multiprogramming," Proc. 22d ACM Nat. Conf. 1967, pp. 135-150.

22. Peter Wegner, Programming Languages, Information Structures, and Machine Organization. New York: McGraw-Hill, 1968.

1. REPORT NO.	2. GOVERNMENT ACCESSION NO.	3. RECIPIENT'S CATALOG NO.

4. TITLE AND SUBTITLE	5. REPORT DATE
C-MOS Array Design Techniques: SUMC Multiprocessor System Study (Contract Modification 7)	May 1972
	6. PERFORMING ORGANIZATION CODE
7. AUTHOR(S)	8. PERFORMING ORGANIZATION REPORT #
W.A. Clapp, W.A. Helbig, A.S. Merriam	
9. PERFORMING ORGANIZATION NAME AND ADDRESS	10. WORK UNIT NO.
Advanced Technology Laboratories Government and Commercial Systems RCA Camden, New Jersey 08102	11. CONTRACT OR GRANT NO. NAS 12-2233
	13. TYPE OF REPORT & PERIOD COVERED Contractor Report
12. SPONSORING AGENCY NAME AND ADDRESS George C. Marshall Space Flight Center National Aeronautics and Space Administration Marshall Space Flight Center, Alabama 35812	14. SPONSORING AGENCY CODE

15. SUPPLEMENTARY NOTES

Work performed for George C. Marshall Space Flight Center Astrionics Laboratory

16. ABSTRACT

The current capabilities of LSI techniques for speed and reliability, plus the possibilities of assembling large configurations of LSI logic and storage elements, have demanded the study of multiprocessors and multiprocessing techniques, problems, and potentialities. The design of a highly reliable and powerful multiprocessor system of small physical size, which results from such studies, is of great interest for long-life space missions.

The present study first evaluates three previous systems studies for the Space Ultrareliable Modular Computer multiprocessing system by IBM, CSC, and Intermetrics, respectively, and then describes a multiprocessing system which originates in the best results of these studies and carries them further both in extent and detail. The proposed system is flexibly configured with up to four central processors, four I/O processors, and 16 main memory units, plus auxiliary memory and peripheral devices. This multiprocessor system features a multilevel interrupt, qualified S/360 compatibility for ground-based generation of programs, virtual memory management of a storage hierarchy through I/O processors, and multiport access to multiple and shared memory units.

In conclusion, the present study is viewed as one in a continuum leading to construction of a multiprocessor system meeting NASA mission objectives; therefore, designation and brief discussion of the most profitable details for simulation and further study have been included.

17. KEY WORDS		18. DISTRIBUTION STATEMENT
Operating Systems	Reconfigurable Systems	
Multiprocessing Systems		
Memory Management	LSI Applications	
Multilevel Interrupts	Modular Processors	
Priority Scheduling		
Information Structures		

19. SECURITY CLASSIF. (of this report)	20. SECURITY CLASSIF. (of this page)	21. NO. OF PAGES	22. PRICE
Unclassified	Unclassified	161	