This paper describes the major elements of a programmed operating system for a complex of five computers employed at the Gemini Mission Control Center. The system was designed for an application environment that includes real-time space missions, simulated real-time exercises, and extensive job-shop operations. Relationships among programs, input/output control, and various operational techniques are described. The characteristics of a statistics-gathering routine are outlined.

Aspects of the Gemini real-time operating system

by J. H. Mueller

Three generations of programming systems have provided real-time ground support for manned spaceflight programs. The first of these was the Mercury system, which became operational in 1960.[1] The third and current system supports the Apollo program. The subject of this paper is the second in the series, the Gemini system, developed for NASA at the Manned Spaceflight Center in Houston, Texas.[2]

The contract for this system specified two real-time systems: one for mission support, and one for vehicle and network simulation. Since these systems were to operate on identical hardware configurations, there seemed no reason why a suitably comprehensive set of control-program services could not serve both needs. Although the resultant system is highly application-oriented, the lessons learned in the course of its design and operation may be of use to others who are developing large-scale real-time operating systems.

The real-time computer complex

The computational facility at the Mission Control Center in Houston is called the Real Time Computer Complex (RTCC).[3,4] The Gemini RTCC configuration, schematically illustrated in Figure 1, consists of five IBM 7094-II computers connected to the System Selector Unit, a plugboard-controlled switching unit that permits the computer subsystems to be configured singly or in combination for various mission-support, simulation, program-testing, and equipment-testing functions of the RTCC.

Each of the 7094-II systems (Figure 2) has 65K words of main memory and provisions for automatic relocation and memory protection.[5] At no degradation of CPU performance, the sixteen-bit relocation and protection registers permit dynamic memory management without concern for the physical separation of the two 32K memory boxes. The relocation method adds the contents of a base register to the effective address immediately prior to memory reference. The eight low-order address bits are ignored, giving a relocation precision of 256 words: consequently, an area dealt with in dynamic memory allocation consists of one or more contiguous 256-word blocks. In the protect mode, references within the upper and lower bounds are permitted, whereas an attempted reference outside the bounds is blocked and causes an interruption. Protection does not distinguish among fetches, stores, or branches. When protection is not in effect, relocation may be applied at the instruction level. This permits the control program to access operands in relocated storage simply by applying the relative displacement in the problem program to the relocation register, *effective for that reference only*.

Each 7094-II has one IBM 2361 Large Capacity Storage (LCS) unit with a 524K-word capacity.[6] Access to the LCS is via the 7286 Channel and the effective transfer rate between the LCS and main storage approaches 256K words per second. Real-time information enters and leaves the system via the 7281 Data Communications Channel (except for television output data, which is transmitted over a direct-data connection at 256K words per second). Data Channels A and B support the card reader, line printer, and tape drives.

The 7281 also provides timing facilities. An interrupt may be generated at a program-specified number of 100-microsecond intervals; concurrently, a synchronized, exact one-second interrupt serves to prevent timing drifts. For very precise timing, each 7094-II is equipped with a 36-bit register that increments at 10-microsecond intervals. This register can be read or reset to zero under program control.

Figure 1 RTCC configuration

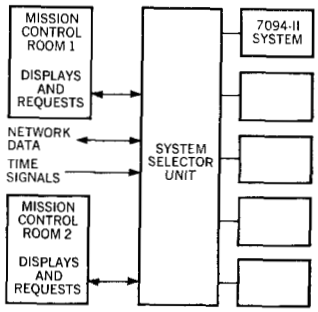


Figure 2 RTCC 7094-II system

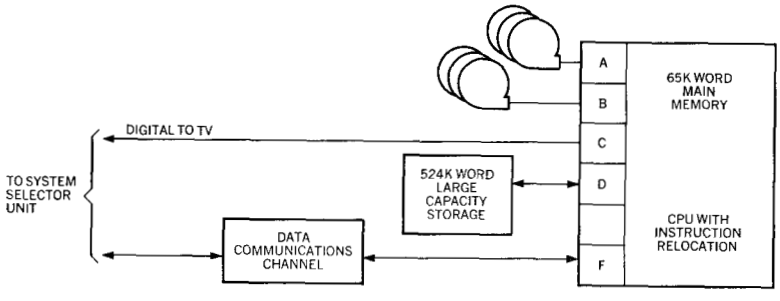
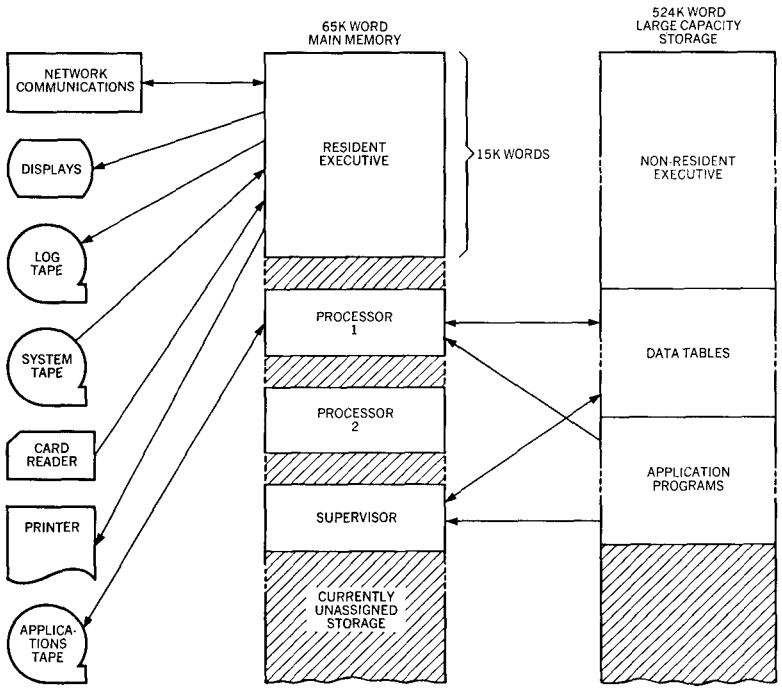


Figure 3 Gemini real-time operating system



Program structure

The Gemini real-time operating systems are composed of the three fundamental elements suggested in Figure 3: the Executive Control Program (to be called "Executive"), application programs, and data tables. "Resident" Executive is held permanently in main storage to service interrupts, allocate core, and perform miscellaneous frequently-used services. The resident Executive and its associated tables occupy about 15K words of main storage. For dynamic display equipment allocation, LCS allocation, error message composition, system tape handling, and similar services, nonresident, or transient modules, are loaded into main storage as needed. These transient modules contend on a priority basis with the application programs for the remaining 50K of main storage.

Application programs are of two types: *supervisors* and *processors*; a supervisor program directs the execution of processor programs. (This unusual terminology seems to have arisen out of historical accidents.) Processors act at explicit directives from supervisors, much as closed subroutines respond to calls from a higher level. Processors are available to all supervisors. An elementary hierarchy might be composed of Launch, Orbit, and Re-entry supervisors that share Data Conversion, Numerical Integration, Differential Correction, and Display Generation processors.

Executive recognizes only two levels of hierarchy: all supervisors are equal, as are all processors. Supervisors may transmit information to one another and may call upon a processor. However, processors cannot call upon one another. A processor simply executes and then returns control to its calling supervisor; moreover, it returns control via Executive because it is unable to identify the calling supervisor. A supervisor or processor may embrace any number of subroutines. A processing sequence involving one supervisor and two processors is illustrated in Figure 4.

Some special transient Executive programs require direct referencing into resident Executive data; consequently they are not subjected to protection, although relocation is still applied. All nonresident Executive modules are structured into processors and supervisors. The relocate and protect scheme prohibits direct referencing or calling between two programs. Consequently, every supervisor-to-supervisor or supervisor-to-processor call requires that the calling program invoke Executive to transmit control. Although this detail is essential to an understanding of the Gemini System, it is convenient to suppress it in describing such operations as "supervisor A called processor R . . ."

In the LCS, a program exists as though it were to be executed from an area starting with memory location zero (00000). Such a program can be placed in any area of main storage; let x denote the address of the first location in the area. When the program receives control, the relocation register contains x, and the program is executed as if it had been loaded in memory location zero.

Since a program is limited to references within itself, i.e., to references within the bounds set by the protection registers, Executive is designed to provide facilities for inter-program communication. The primary means for communication is a ten-word table or "buffer," called XTRANS, within each program. Before a program receives control, its XTRANS buffer is filled by Executive. Whenever a program requires the service of another program, the calling program fills its own XTRANS with programmer-defined parameters.

Each supervisor or processor is represented by an entry in the Executive priority table. An entry reflects program status in terms of the following information:

- program is (is not) currently operating
- work is (is not) defined in the program queue
- program is (is not) in main storage
- program is currently being loaded into main storage
- program is available in LCS
- program is a supervisor (processor)
- program is an Executive extension that can run without protection

The order of the entries in the priority table is established prior to initialization. During execution, when an XTRANS is created for a program, an entry for this program is chained in priority order into an "active" list. Executive scans the active chain from the top

Figure 4 A supervisor-processor sequence

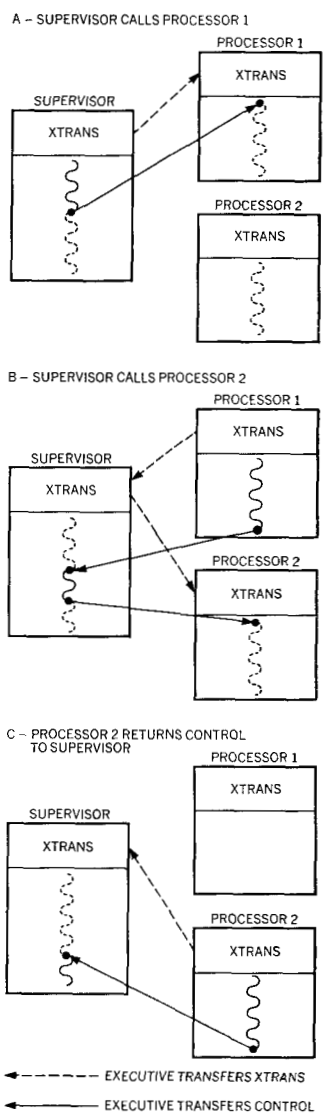
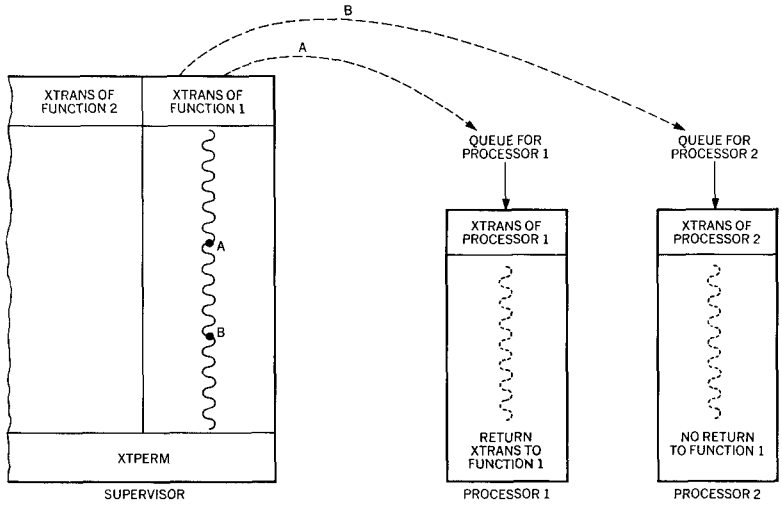


Figure 5 Calls sent to processors



(highest priority) each time a status change occurs. Whenever Executive discovers a program ready for execution, the scan stops and that program receives control.

supervisor structure

We have introduced a supervisor as though it were a single program. This is an over-simplification; actually, a supervisor can consist of one or more programs called supervisor "functions." Each function has its own XTRANS area, but all share a common XTPERM. Each supervisor has one entry in the Executive priority table; each function has a second-level priority and contends for control within its supervisor.

As a simple example of a multi-function supervisor, a supervisor dedicated to the processing of launch data may contain one function for each class of data. The function concept is intended to endow the supervisor with a capability for generating concurrent requests of processors. Multiprogramming can occur almost without the application programmer's being aware of it. With a number of concurrent requests for processing, the Executive can attempt to maximize the effective utilization of the CPU.

The supervisor function can call a processor in several different ways. One method uses a processor as a closed subroutine with control returning to the function at the point immediately following the call. In this method, the function is out of operation until the processor returns control. An alternative approach allows the function to *send* a call to a processor (Figure 5). The function gives up control only while Executive enqueues the XTRANS for the specified supervisor. In this manner, a function may sequentially call a number of processors before any of the called processors begins operating. A call having been *sent* to a processor, the processor is given the option of determining whether or not a return is to be made. If a return is not intended, the processor simply represents

an "orphan" task in multiprogramming; when the processor completes, the task completes and disappears. In a typical case, a function calls a processor to update a specific display. The function expects no return unless something unexpected develops in the processor's computations. If the unexpected occurs, the function will receive control at its entry point—at which time its XTRANS will contain information from the processor's XTRANS.

system data

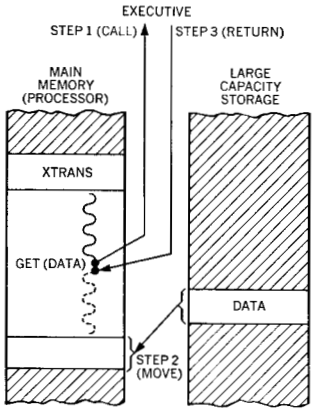
For data it must access frequently, each supervisor contains a private table called an XTPERM. Whenever the main storage occupied by a supervisor must be made available for other uses, the Executive preserves the affected XTPERM. At the time the supervisor again receives control, its XTPERM is restored. The size of an XTPERM is established by the programmer who designs the supervisor. As a simple example, consider a supervisor that needs to know the elapsed time since its most recent execution. Assume that at each instance the supervisor receives control, the second word of XTRANS contains the current time. This supervisor would compute the elapsed time by subtracting the previous entry time (saved in XTPERM) from the new time in XTRANS, and then update the entry in XTPERM with the new time from XTRANS.

Although XTPERM has many uses, a facility for handling much larger quantities of data is provided by "data tables." A data table is dimensioned prior to run initialization and is dimensionally static during a run. It is unrelated to any program, except on a dynamic basis, i.e., data table information belongs to the *system*. Executive help is required in reading from or writing into a data table because table reference generally involves input/output operations.

Data tables are meant to be "device independent" in the sense that a programmer need not know which device contains a data table. The conventional methods of record processing are supported by the Executive IOCS services—a data table may be composed of one or more record blocks. However, the data tables may be used, and are commonly used, as simple extensions of core storage data areas or, in the FORTRAN idiom, "bufferable common." The programmer who has been using a data table assigned to tape is confronted with no difficulties if that data table is reassigned to the LCS. However, the programmer using the word-level referencing permitted by LCS finds himself "unsupported" if the data table is reassigned to tape. In RTCC practice, almost all Z-tables are located in the LCS. The fact that system data and, generally, all of the programs needed for the current processing phase, are available in LCS permits operation with a main storage significantly smaller than would be necessary with conventional auxiliary devices, such as drums or disks. Moreover, programs may read or write individual words of the data table.

The use of a data table can be illustrated with a simple example of a typical input processing cycle. Such a cycle entails preliminary computation on the raw input data, merging of new and old data, extensive computation on combined data, and updating operations that propagate the effect, if any, of the new data throughout the

Figure 6 Processor getting data from LCS



system. The cycle is accompanied by frequent real-time inputs, but relatively little data. (Until the recent capacity increases in telemetry rates, RTCC real-time inputs rarely exceeded 1K words per second, and much of that was redundant. The LCS data tables and programs produced the real I/O load, frequently exceeding 100K words per second.)

Throughout a cycle, whenever a program needs "old" data, the Real Time Input/Output Control System (RTIOCS) brings the information from LCS (Figure 6). Whenever a program must update information in a data table, the RTIOCS writes the data into the specified data table. An application program simply calls Executive with the name of the data table, the location in main storage from which data is to be written or into which data is to be read, and the amount of data involved.

To support a reference to an RTIOCS service, one or more small Executive subroutines are included in the user's program when the system tape is constructed. During execution, a call upon RTIOCS gives control to the appended subroutine. The subroutine executes a Store-and-Trap instruction (STR), where the relocation mode is turned off and control passes to the resident Executive. The STR instruction is used for all requests for Executive support; a code in the STR identifies the service required.

Normally, a program requesting data-table service is not restarted until the service operation has been completed. In some cases involving very slow output devices, RTIOCS provides some buffering, and control is returned to the program before the physical operation has been completed. On the other hand, for the usual data-table operation involving LCS, the program simply waits until the operation has been completed.

Programmers are given no control over buffering. Design discussions all ultimately led to the same conclusions: (1) Executive could be equipped to decide when and how much buffering was required, and (2) most application programmers prefer to leave this complication to the control system. The decision was also consistent with a decision to use FORTRAN. Approximately half of the code in the manned-support systems was written in FORTRAN; every Executive service is FORTRAN compatible and invokable by the standard CALL statement. The XTPERM facility of the supervisors could easily have been simulated by data tables. The two approaches reflect a trade-off between time and space; the more frequently data is referenced by a supervisor, the more likely the data is to be kept in the XTPERM; the greater the space needed, the more likely the data is to be kept in a data table. Also, data in XTPERM are private to a supervisor, whereas data tables are not.

Since the names of data tables can be handled as data, there is virtually no limit to the amount of data a processor can access from the information supplied in XTRANS. In fact, many of the trajectory processors for the Gemini-Agena mission systems operate on data from tables named by the calling supervisors, but which do not appear in the processor's program.

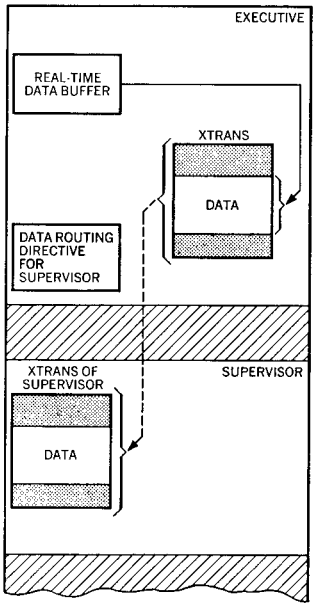
data routing

The process by which Executive delivers real-time data to the application programs is termed "routing." Consider a supervisor that must produce display output every second. This supervisor instructs Executive to give it control at every second. The Executive then files the request in a "routing directive" that remains operative until modified or cancelled. Every second thereafter, while scanning its time-routing directives, the Executive creates a distinctively identified XTRANS and immediately attempts to give the XTRANS to the specified supervisor. But this is not always possible; the supervisor may not be in main storage, it may currently be operating, or work of higher priority may have intervened. In such an event, the XTRANS is placed in a first-in, first-out queue for the program. When a resident program with work in its queue receives the highest active priority, the XTRANS is moved into the program and control passes from Executive to that program.

The data-routing process is similar to the routing of timing information. A data-routing directive issued by an application program is of two parts: the first identifies the data to be saved; the second defines the action to be taken when designated data have entered the system. Two options are available: "direct routing" and "store routing."

Direct routing takes advantage of the fact that RTCC has few variable-length messages. Some messages are small enough to fit within an XTRANS: for these, the programmer usually specifies direct routing (Figure 7). When the interrupt occurs, Executive simply creates an XTRANS block, dumps the data into it, and places this XTRANS into the queue for the program named in the routing directive. When the program reaches this XTRANS, the data are immediately available for processing. If the message exceeds the XTRANS space, Executive places it in an Executive buffer and stores the buffer address in XTRANS. Using the information in the XTRANS, the program obtains the message by calling upon RTIOCS to move the message into the program's area.

Figure 7 Direct-mode routing



Real-time processing usually involves a collection phase followed by a processing phase, and frequently these phases overlap. Data collection is handled by Executive in response to a routing directive that identifies the data and requests that the data be placed in a specified data table. A separate routing directive causes Executive to generate a periodic XTRANS as a function of time. When the designated program receives this XTRANS, the RTIOCS is called to read the collected data from the data table.

Store-mode routing provides an interesting use of a particular RTIOCS service. A data table can be defined as "circular," in which case data are stored sequentially and overflow data displace data at the beginning of the table. Hence, the contents of the table are always that most recently received. This feature has proved useful in periods of heavy loads. Data collection proceeds continuously as real-time data arrives. It is desirable to process all input data, but when the system is heavily loaded, less frequent processing of inputs is acceptable. Under heavy loads, more important work may

displace some of the input processing. When the load decreases, processing of input resumes with the latest data.

I/O control

The goals for the design of the RTIOCS were convenience and simplicity. Basically, the user has two RTIOCS calls: GET and PUT. Each GET or PUT is defined by one or more "argument sets." An argument set must contain three parameters and may have as many as five, namely,

- data table name
- internal buffer location
- word count
- block or table subscript (optional)
- block identifier (optional).

The latter two parameters apply to blocked data tables. A data table of 500 words may be defined as one 500-word block, five 100-word blocks, 100 five-words blocks, etc. If ZNAMEX is defined as two 250-words blocks, the following call

CALL GET (ZNAMEX, internal buffer, 2, 3, 2)

will yield the 253rd and 254th words of the table. However, if ZNAMEX were defined as 100 five-word blocks, the call would "get" the eighth and ninth words of the table. In system creation, the data-table name is transformed into a pointer to a control block within Executive. Control-block information is used to allocate the data table to a physical device at initialization time and to translate GET's and PUT's during execution. Each I/O device supported by the Executive is represented by one or more control blocks. Control blocks are used in the allocation of main storage, LCS, magnetic tapes, the on-line printer, the direct data-to-TV connection, and the subchannels of the 7281.

Operational considerations

Virtually all of the 7094 computing at the RTCC operates either under Executive or GAJOB (Gemini-Apollo IBJOB), a modified IBSYS/IBJOB system.[7] With FORTRAN and IBMAP, GAJOB is used to build and test programs. The Editor in GAJOB provides master-library and system-tape services. All real-time systems have binary and symbolic master files. With the Editor, decks may be altered against the binary or symbolic masters, and individual source cards may be altered against the symbolic master. Although there are various modes and methods of testing programs, subsystems, and full real-time systems, it is significant that the programs being tested need never be changed to satisfy the testing environment.

Since all the Executive services are provided via the CALL statement, a simulator of the real-time environment was easily provided under GAJOB by means of library subroutines. This capability permits the testing of a single processor or supervisor, or of a supervisor and several processors, in a batched-job system. Some of the special features of Executive's multiprogramming facility cannot

be simulated adequately in a sequential environment, but for most unit, string, or subsystem testing, the simulator is very effective. The fact that processing is sequential often makes it possible to identify programming mistakes before the environment changes completely (a constant problem in multiprogramming debugging). Moreover, a capability for significant debugging in a batch mode greatly economizes the computer time required for delivery of checked-out systems.

When unit, string, and subsystem tests have been completed, the programs can move unchanged into the real-time system testing environment, the first step being to create a real-time system tape.

The Executive has been described as a single control program serving many real-time applications systems. The linkages between the Executive and the application system are resolved by the Editor. The user defines his application system for the Executive by building tables in the Executive. This is accomplished by inserting macrostatements into a special Executive deck. When this deck is assembled, the user's section of Executive is created. Basically, these macrostatements define:

- Application program priorities
- File control blocks (for z-tables)
- Initial routing directives

The Editor combines this user's deck with the Executive code and produces the Executive nucleus. In the process, all symbolic names (program names, z-table names, etc.,) are translated into indexes. Essentially, the translation trades pre-execution time to avoid translation in real time. The Editor writes the nucleus on the real-time system tape and copies the remainder of non-resident Executive and the application programs. As a non-nucleus program is placed on the real-time system tape, each reference to the translated symbols is replaced by that symbol's assigned index. Thereafter, corrections may be made to programs on the tape at little expense, provided no changes are made to the nucleus indexes on the tape.

After the real-time system tape is loaded, the monitor takes control, loads the LCS with z-tables and programs, and performs machine diagnostics. The user can then select among a variety of options. The first significant option is whether or not the user requires clock synchronization. Unless external devices (including other computers and/or people) are involved, synchronized time is rarely used. Unsynchronized or simulated time uses an internal clock; when the computer becomes idle, this clock is set forward to the next clock interrupt. Using this feature, an orbit of 90 minutes is often simulated in a few minutes without changes to the orbit program. In fact, there is no way an application program can tell that a simulated clock is being used.

When running in the simulation mode, the user can specify that any or all real-time inputs be simulated with fabricated data from

one device. Real-time inputs can be simulated, accept live data, or go unused—in any combination. Furthermore, the real-time output devices can be used or their outputs diverted (by modifying the file control block) to the LCS. It should be noted that only when all input is under its control can the simulation monitor guarantee that input data will appear exactly as it would in a similar synchronized real-time run. However, for most runs that combine real-time and simulated devices, this restriction presents no problems.

The RTCC version of the IBJOB debugging system, heavily used in job-shop runs, is also available when running a real-time system in the simulation mode. The debugging package operates with the simulated clock turned off, so that the application programs cannot recognize that the debugging operations are taking place. When a synchronized, or true, real-time run is specified, initialization automatically removes any debugging requests.

For real-time runs, a Statistics Gathering System (SGS) is optional. This option applies only to the synchronized mode and generally requires about 5 percent of the CPU time in overhead. (The RTCC experience has been that if 5 percent is the difference between success and failure in a real-time run, there really is no difference.) Statistics are accumulated in three categories:

- Internal Executive Logic—frequency of use, average execution time, allocation attempts and successes, etc.
- Supervisor and Processor—number of uses, average execution time, number of executions per fetch from LCS, number of Executive CALL's, etc.
- CPU Utilization—total execution time, time for waiting for I/O, and idle time

The SGS recording routine is part of resident Executive. SGS is activated through branches inserted at selected locations in Executive. Thereafter, when the location counter reaches one of the locations, control passes to the recording routine, which updates the appropriate SGS tables. Periodically, transient low-priority SGS programs produce output from the tables. Originally conceived to support the extensive GPSS modeling activities of RTCC, the SGS option has proved useful to many of the programmers as a tool for analyzing their systems.

Summary comment

The Gemini system became operational in early 1964; it supported the remainder of the Gemini project and the 1966 Apollo missions. Its simplicity permitted several hundred programmers—of wide experience ranges and many with no prior experience at all—to learn and apply the Gemini concepts and facilities in a short time.

The use of FORTRAN facilitated rapid responses to NASA's constantly changing requirements—change is the norm in a developmental project such as manned spaceflight. The routing and data-

table facilities have been expanded and incorporated into the Gemini System's successor.

Finally, without its SGS measurement tools, the programming system could not have reached the desired performance levels, and its designers would have had less self-assurance regarding the success of the system.

ACKNOWLEDGMENTS

While the RTCC system described in this paper represents the efforts of many people in both design and implementation, the author particularly wishes to acknowledge the contributions of Mr. Charles T. Mullins for many of the initial design concepts and Mr. Robert L. Hoffman, who managed the system design, analysis, and implementation activities from mid-Mercury to Apollo.

CITED REFERENCES AND FOOTNOTES

1. "Project Mercury real-time computational and data-flow system."
   S. I. Gass, "The role of digital computers in Project Mercury," *AFIPS Conference Proceedings, Eastern Joint Computer Conference,* Macmillan Co., New York, 33–46 (December 1961).
   M. B. Scott and R. Hoffman, "The Mercury programming system," *ibid.,* 47–53.
   W. K. Green and A. Peckar, "Real-time simulation in Project Mercury," *ibid.,* 54–65.
   R. D. Peavey and J. E. Hamlin, "Project Mercury launch monitor subsystem (LMSS)," *ibid.,* 66–78.
2. J. E. Hamlin, "A general description of the National Aeronautics and Space Administration Real-Time Computing Complex located at the Manned Spacecraft Center, Houston, Texas, "*Association for Computing Machinery, Proceedings of the 19th National Conference,* Philadelphia, Pennsylvania, A2.2-1–A2.2-22, (August 1964).
3. A. M. Pfaff and C. C. Rawlins, "NASA-MSC's real-time computer complex," *Proceedings of the Real-Time Systems Seminar,* Houston, Texas, International Business Machines Corporation (November 1966).
4. W. S. Harner, J. F. Dinwoodie, and R. D. Compenni, "The role of real-time computer complexes in support of manned spaceflight," *ibid.*
   The Real Time Computer Complex gathers and processes data and generates displays for all non-military manned spaceflights. During mission support, radar tracking and telemetry information are the major inputs. The major computations performed are trajectory analysis and prediction, recommended powered thrusts and maneuvers, generation of digital commands for transmission to the spacecraft, logging of events, and maintenance of continuous displays of a variety of data categories for flight controllers.
   Although mission support is the major element determining the RTCC configuration, it accounts for a small fraction of total computer utilization. Other computational activities include spaceflight simulation, astronaut and flight controller training, and development and testing of programs to support all Mission Control Center activities.
   The programming effort at RTCC involves two classes of programs: the operating-system programs, and the application programs designed to run under operating system control. Program development and testing involves non-real-time batch processing. Successful assembly or compilation does not yield mission-ready program systems; extensive program testing is nor-

mally required, and this testing in turn requires special programs and equipment.

5. The basic relocation hardware package was derived from MIT's Project MAC. The 65K extensions and the capability to invoke or ignore relocation on an individual instruction basis were the sole creations of Mrs. Jane G. Jodeit.
6. T. A. Humphrey, "Large core storage utilization in theory and in practice," *AFIPS Conference Proceedings, Spring Joint Computer Conference* **30,** Academic Press, 719–727 (April 1967).
7. "Design of an integrated programming and operating system."
   A. S. Noble, Jr., "Part I, System considerations and the monitor," *IBM Systems Journal* **2,** 153–161 (June 1963).
   R. B. Talmadge, "Part II, The assembly program and its language," *ibid.*, 162–179.
   R. Hedberg, "Part III, The expanded function of the loader," *IBM Systems Journal* **2,** 298–310 (September-December 1963).