intel®

ARTICLE REPRINT

AR-125

April 1980

Modular Multitasking Executive Cuts Cost of 16-Bit OS Design

Joseph Harakal

Electronic Design, March 15, 1980

Modular multitasking executive cuts cost of 16-bit-OS design

A modular, real-time multitasking operating system for single-board computers allows custom operating systems to be assembled largely from off-the-shelf software components. Such systems, when needed in OEM single-board μC applications, pose problems—rarely can an OEM afford man-years of effort to develop the intimate familiarity with the hardware that's needed to design executive software. But with Intel's RMX/86 system for iSBC 86 single-board computers, he won't have to.

In addition, this second-generation, 16-bit system added error-handling, flexible command-line decode, and other advanced OS capabilities to previous options available on the older RMX/80.

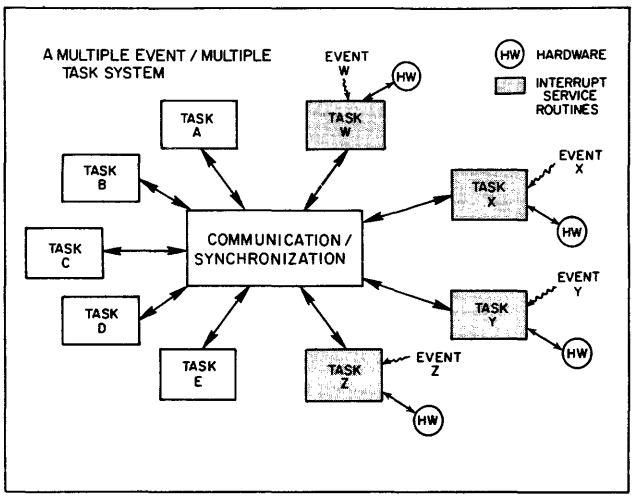
All real-time multitasking systems require executive software not only to manage the resources shared by the task programs (CPU time, memory and I/O), but also respond to interrupts and then allocate the resources according to established priorities. Normally, these functions are intermingled in an OS with higher-level system functions that often prove superfluous in single-board computer applications.

The RMX/86 OS package combines all those required executive functions in a single module, called the nucleus. Other modules in the package tailor the executive system to its application by adding higher-level OS functions such as disk-file systems. The task programs and optional modules connect to the nucleus with simple software interfaces. Users can also add their own extensions.

Since the nucleus is essentially open-ended, it can serve as the software foundation for expanding both the operating system and the variety of task programs. Although most single-board computer applications are dedicated, many do require higher-level capabilities.

The OEM way

The modular approach to system software fits in with the way most OEMs (and high-volume end users) apply single-board computers. They generally start with a minimum amount of hardware (often, just a



1. In a real-time multitasking system, task modules (A through E) can often perform their functions only after hardware-generated interrupts are serviced (highlighted). The executive in such a system provides intertask communications and synchronization.

RMX/80 vs RMX/86 features

RMX/80	RMX/86
Nuclei	**Nuclei**
For iSBC 80/10	One serves all iSBC boards
For iSBC 80/20	Free-space manager
For iSBC 80/30	Exceptional-conditions handler
Optional modules	**Optional modules**
Disk-file system	I/O system
Disk I/O	Hierarchical file system
Terminal handlers	Numbered file system
Free space manager	Internal file system
Analog I/O handlers	Physical file system
Bootstrap loader	Interfaces for custom files and I/O
Debuggers	Human interface system
Support packages	Command-line decoder
Fortran-80 run-time	
Basic-80 interpreter	
8080/8085 fundamental support	

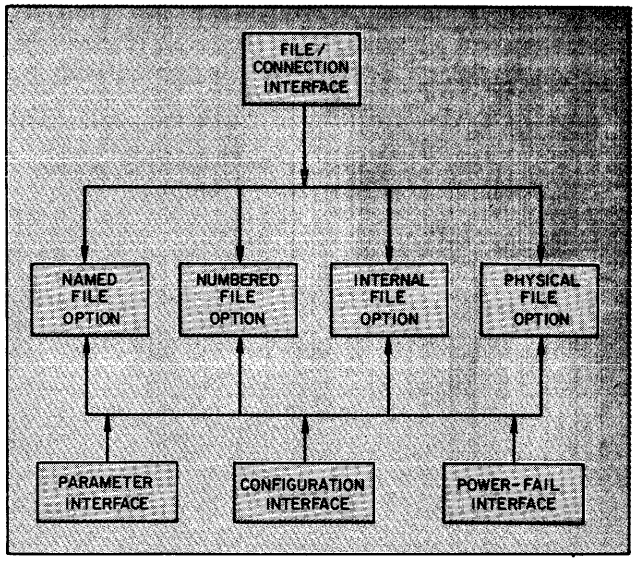
Joseph Harakal, Software Product Manager, Intel Corp., 5200 Elam Young Pkwy., Hillsboro, OR 97123.

AFN-01435A

single board) to begin an application at low cost. Then, when users are satisfied with the original functions and are willing to buy more, the OEM adds the executive options, tasks and hardware—with as fast a turnaround as possible.

The problem with conventional "general-purpose" software systems is that they usually depend on hardware that the application may not need—for example, standard peripherals whereas a typical OEM system uses special peripherals. Modular OSs are designed to accommodate both.

What's more, a conventional system can make it difficult and/or awkward to use new peripherals or new technology, such as magnetic-bubble memory—most command-line decoders, for instance, are not accessible to the user. So, a user may discover that there is no straightforward way of adding new facilities.



2. The RMX/86 operating system treats all I/O as files—a feature that makes it easy to add new peripherals and special files.

Call it foundation software

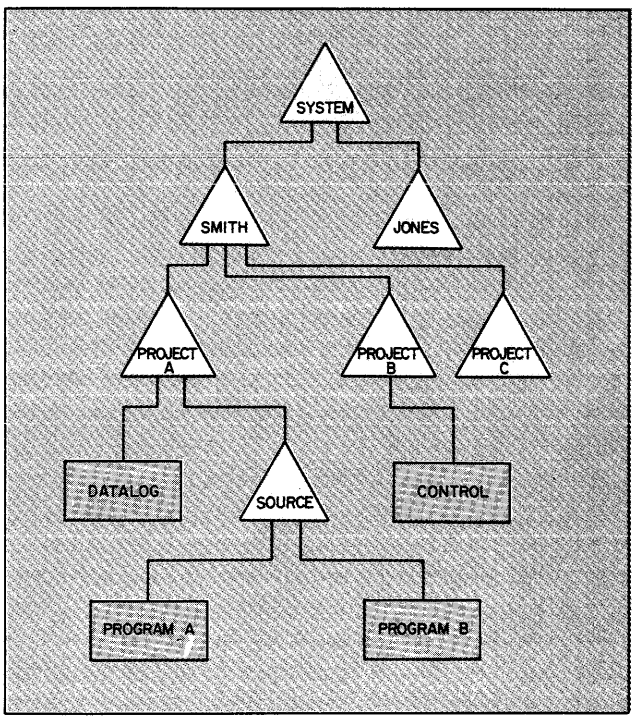
For an even closer fit with single-board computer applications, the RMX/86 modules closely parallel hardware modularity: Each computer board contains program and data memory, serial and parallel I/O, and other generally required functions in addition to the CPU. Each user's system is expandable with optional modules. Frequently used devices like disk controllers and analog I/O are available. In addition, the user can connect custom devices to his system via the Multibus architecture.

Corresponding to the hardware, systems software manages CPU, memory and I/O resources. Linked to optional modules, it can support standard iSBC devices like consoles and disk controllers. Similarly, users may add device-driver software modules for their custom peripherals. All software can reside in EPROM/ROM if mass storage is not available; otherwise, most of the system can be disk-resident. The disk-file module is suitable for such applications as data logging.

The RMX/86 is designed for configuration on an Intellec development system according to user requirements. The same system supports task-module development as well as linking and locating in both high-level and assembly languages. It also provides libraries of frequently used program functions to minimize the amount of code the system designer must write and debug.

To see how the RMX/86 works, consider a typical example. Say an OEM develops a factory heating and air-conditioning control system having a single-board computer, analog I/O, special control devices and an operator terminal. He uses the nucleus, analog-handler and terminal-handler modules, plus user tasks stored in EPROM.

But suppose the OEM's customer wants to enhance the system with, say, disk storage. He simply adds a disk controller board and uses I/O system software. The original programs could also be made disk-resident. If the original application had been based on a



3. A hierarchical file system eliminates the need for scanning all the files on a disk. If Smith is working on Project A, he only has to choose between the files related to his project.

conventional executive, extensive redevelopment would have been necessary.

If, on the other hand, the application had used the full I/O system from the start, initial sales would have suffered because the OEM's system would have been more costly. And if the software is not extensible, future sales opportunities are lost.

A real-time multitasking executive gives a program the means to monitor and control external events. Tasks run concurrently, using communications and synchronization services of the executive (Fig. 1). Events are signaled with interrupts, and the executive schedules resources on the basis of priority—for example, a task trying to bring a factory under control should have a very high priority. The executive decides whether a running task should be interrupted to process data from an interrupting device.

The RMX/86 nucleus makes such event-driven priority scheduling happen through resource management. It monitors system states, determines task requirements, allocates resources and gathers them for reallocation. Resources include CPU time, memory and I/O.

As in other systems, the highest-priority task that's ready to run uses the CPU; others are put on a read list. Finishing the high interrupt, the nucleus returns the CPU to the highest-priority ready task.

Managing inner space

A free-space manager, built into the nucleus, handles the 8086's megabyte addressing range, making sure that memory is used efficiently. The manager organizes memory into a tree-structured hierarchy of pools according to job requirements, and returns memory to pools. When the nucleus requires memory for a job, mailboxes or other functions, the manager provides it. Or, when a task requests memory—for example, to input data—the manager allocates it. This is done in segments that are multiples of 16 bytes, corresponding to the 8086's segmented memory.

Tasks send data to each other through mailboxes which contain messages located in RAM. As in other systems, separate mailboxes are used for receiving and for responding.

In the RMX/86, however, mailboxes provide several options, including synchronization, mutual exclusion (which prevents one task from destroying another task's data) and communications—for example, with the outside world—through modules such as the terminal handler.

The RMX/86 nucleus also provides other means for communications. One example is semaphores—low-overhead mechanisms for synchronization operations, resource allocation and mutual exclusion that require a simple flag. For example, one task can simply set a flag to tell another task that an event has happened ("Analog data received").

All these functions are accessible with simple calls to the nucleus. For example, just two calls are needed to use the free-space manager: CREATE-SEGMENT to request memory and DELETE-SEGMENT to relinquish memory. Likewise, to obtain an mailbox, the task simply names the mailbox desired. The programmer doesn't need to know the internal structure of the executive, since the functions are accessible through easily programmed interfaces.

Errors, big and small

Another RMX/86 feature, the exceptional-condition handling, makes error handling selective rather than all or nothing. Programs can be designed to manage error conditions and take corrective action.

Modular multitasking comes on

Multitasking design is coming to the fore, especially for single-board-μC applications that usually require a lot more software than previous μCs—an advance from simple foreground-background programs to techniques based on event priorities.

Although single-board μCs started out in the mid-1970s at the low end of the OEM performance range, they have now reached the top in performance and memory capacity. As more and more OEMs and users take advantage of that increased capability, the size of applications programs grows—and grows.

In the same period, the costs for developing software, salaries and overhead have almost doubled. Moreover, skilled programmers have become one of the industry's most limited resources. No wonder that software costs comprise up to 80% of system development costs today, and that the emphasis has shifted from in-house software design to buying off-the-shelf programs.

Because multitasking designs have to be highly modular, time-saving tools such as high-level languages, program libraries and off-the-shelf software can be used freely to help keep development, maintenance and expansion costs under control. Code written in high-level languages is a bargain today, compared to code written in assembly language: around $2.50 a byte vs $10 for assembly language. High-level code is not as compact, but it's far more cost-effective for the 80% of the tasks that run only about 20% of the time in typical applications.

Today, there's a growing choice of languages. Structured languages like PL/M and Pascal fit well into the "top-down" modular design technique used to divide an application into tasks. Others, like Fortran for mathematical applications and Basic for easy end-user programming, are also available.

In general, a real-time multitasking executive offers a reasonable choice for the user who has a lot of software to write, must meet special requirements, and has no time to develop a custom operating system. The RMX/86 system, with its modular design, fills the bill to save development cost.

AFN-01435A

First, there's an option to specify to the nucleus whether or not there should be any error handling for a particular task. A programmer can write his own handler either to abort a task or to program a specific course of action—for example, report exceptional conditions and continue with next instruction; load copy of module and try again; start alarm program.

The exceptional-conditions support also detects programming errors such as a wrong call to the nucleus and system problems like insufficient memory. Naturally, the RMX/86 provides all normal OS functions. System options include a terminal handler for CRT and TTY consoles device drivers for Intel's floppy and hard-disk controllers and an integrated I/O system. A subsystem of the I/O system supports tree-structured directories hierarchical named files (Fig. 2).

I/O features are vital

Most single-board computers are used with special peripheral devices, and many with other kinds of files and media. So, the I/O system is designed to make it easier to add special files, new peripherals and custom device drivers—the user need never feel locked in.

The RMX/86 I/O system provides the user with a very general file concept—as a data sink or source. The characteristics of a specific storage medium dictate the access techniques for a given file. For example, a disk file may be accessed either in sequential or random fashion, while a file accessed over a serial link (USART) must be processed serially.

Using the data sink/source concept, the user can develop application programs without worrying about the physical device where the data will be stored. Such device independence simplifies application programming, and existing programs can be used with many devices.

The RMX/86 I/O system supports three types of files.

Physical files represent the lowest interface level to retain device-independent characteristics. They provide a simple, consistent interface to all device drivers. OPEN, CLOSE, READ, WRITE, SEEK and special instructions perform all desired I/O operations.

Stream files provide a temporary data-transmission path between tasks. One task may write data to the stream file while another reads them. The I/O system performs the required synchronization and buffering. Stream files offer the user a simple mechanism for passing data between tasks—one that remains consistent with other file options. The user can simulate an external device while waiting for hardware to be built.

Named files are used for the conventional data storage on mass-storage devices like floppy or hard

```
1       TASKB1: PROCEDURE PUBLIC;

2           call rqend$initstask;
2           CALL INIT;
2           do forever;

3               msg$token=rqreceive$message(mailbox$x,
    -   0FFFFH,@resp$ex,@ex$val);
3               call rq$send$message(rq$normal$th$out,
    -   msg$token,out$resp,@ex$val);
3               msg$token= rq$receive$message(out$resp
    -   ,0FFFFH,@resp$ex,@ex$val);
3               call ro$delete$segment(msg$token,@ex$v
    -   al);

3           end; /* of do forever */
2       end;
```

4. RMX/86 can easily be expanded with user-coded tasks. The one shown here initializes a user program by calling the procedure INIT and helps display messages. RQRECEIVE$MESSAGE is a system primitive that examines the mailbox to be serviced and places the token for the first message there (MSG$TOKEN). Another system primitive, RQ$SENDMESSAGE, puts the token for the message into the terminal handler's output mailbox. The primitive RQ$DELETE$SEGMENT clears the used memory and returns it to the free-memory manager.

A postgraduate system

The lessons learned since 1977 about OEM requirements for executives have fueled the evolution of the RMX/80 system. This has led to the powerful new features of the RMX/86 system.

The RMX/80 began with a nucleus for the first single-board computer (iSBC 80/10). Subsequent boards contained more memory and offered higher throughput. Nuclei for the three later boards were made interchangeable, to act as the "software bus" for transporting applications software from one board to another. The RMX/86 solution is simpler: The new nucleus is hardware-independent so it can be used on future as well as current iSBC 86 boards.

So far, process-control designers have generally refused to add user-programming facilities, to assure that users do not interfere inadvertently with tasks handling critical process conditions. If a program dies during a chemical reaction, for example, a whole batch can be ruined and the processing facilities may have to be flushed out.

The RMX/86 system, however, incorporates facilities for keeping programs alive. Its nucleus contains an exceptional-conditions support. Actually, a powerful error-processing subsystem, it allows single-board computers to be made fault-tolerant with no adverse impact on throughput.

disks for later access by another system. However, the RMX/86 goes one step further by setting up a hierarchical directory of files. This feature lets the user organize his files to be consistent with his application (Fig. 3).

The named-file system has another advantage: It permits file-access checking, so a user can decide which of his files he wants to protect and which to share with other users.

Each of the file options can be configured independently; the user may select the features he needs—neither more nor less. Furthermore, the user can add his own device drivers to the I/O system.

The RMX/86 is designed to offer the user a wide spectrum of convenient functions (Fig 4). For example, the user of mass-storage systems has a display directory and copy files available. Or, the OEM who needs his own interactive capability can easily extend the system's human interface routines to meet his requirements.

As μP applications expand, so does the need for loaders that allow parts of the applications software to reside on disks. The RMX/86 package provides a resident system loader that permits loading for either absolute or relocatable format.■■

Printed in U.S.A./B-206/10K/0480/BA/DD
AFN-01435A